IBM System/3
RPG II Additional Topics
Programmer's Guide

Program Numbers:
5702-RG1 (Model 10)
5704-RG1 (Model 15)
5704-RG2 (Model 15)
5705-RG1 (Model 12)

Third Edition (July 1974)

This is a major revision of, and obsoletes, GC21-7567-1. Information concerning IBM System/3 Model 15 has been added and numerous corrections have been made. Changes to text and illustrations are indicated by a vertical line to the left of the change; new or extensively revised illustrations are denoted by the symbol ● at the left of the caption.

This edition applies to the following IBM System/3 RPG II program products:

Version	Modification	Program Number	System/3 Model
15	00	5702-RG1	8 and 10 Disk
6	00	5704-RG1	15A, B, C
2	00	5704-RG2	15D
4	00	5705-RG1	12

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest *IBM System/3 Bibliography*, GC20-8080, for the editions that are applicable and current.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Use this publication only for the purposes stated in the *Preface*.

Publications are not stocked at the address below. Requests for copies of IBM publications and for technical information about the system should be made to your IBM representative or to the IBM branch office serving your locality.

This publication could contain technical inaccuracies or typographical errors. Use the Reader's Comment Form at the back of this publication to make comments about this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901. Comments become the property of IBM.

Technical Newsletter

This Newsletter No.	GN21-5709
Date	21 December 1979
Base Publication No.	GC21-7567-2
File No.	S3-28
Previous Newsletters	GN21-5616 GN21-5389

IBM System/3
RPG II Additional Topics
Programmer's Guide

This technical newsletter applies to current versions and modifications of the applicable System/3 programs listed in the edition notice, and provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are:

v through viii (text rearranged)
3-7, 3-8
3-17, 3-18
5-65, 5-66
5-75, 5-76
6-33, 6-34
6-37, 6-38
6-47, 6-48
7-7 through 7-10
7-19, 7-20
7-31, 7-32
9-49, 9-50
10-41, 10-42

Changes to text and illustrations are indicated by a vertical line at the left of the change.

Summary of Amendments

Miscellaneous technical changes

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

IBM Technical Newsletter

This Newsletter No. GN21-5616
Date 30 June 1978

Base Publication No. GC21-7567-2
File No. S3-28

Previous Newsletters GN21-5389

IBM System/3
RPG II Additional Topics
Programmer's Guide

This technical newsletter applies to the current versions and modifications of the applicable System/3 programs listed in the edition notice and provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are:

Cover
Title Page, Edition Notice
v, vi
3-11 through 3-16
4-3, 4-4
4-27, 4-28
5-53 through 5-56
5-56.1 through 5-56.6 (added to accommodate new and moved text)
5-77, 5-78

Changes to text and illustrations are indicated by a vertical line at the left of the change.

Summary of Amendments

- Expanded description of the RPG II Linkage Editor, page 5-54
- Included miscellaneous technical changes

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

IBM Technical Newsletter

This Newsletter No.	GN21-5389
Date	24 May 1976
Base Publication No.	GC21-7567-2
File No.	S3-28
Previous Newsletters	None

IBM System/3
RPG II Additional Topics
Programmer's Guide

This technical newsletter provides replacement pages for the subject publication. Pages to be inserted and/or removed are:

i, ii
1-5, 1-6
3-21, 3-22
3-27 through 3-32
3-39, 3-40
4-3, 4-4
4-7, 4-8
5-5, 5-6
5-9, 5-10
5-49, 5-50
5-67, 5-68
5-77 through 5-80
6-9, 6-10
7-9, 7-10
7-19 through 7-22
8-27, 8-28
9-1, 9-2
9-27, 9-28
9-41 through 9-44
9-49, 9-50
9-55, 9-56
10-11 through 10-14
10-21 through 10-24

Changes to text and illustrations are indicated by a vertical line at the left of the change; new or extensively revised illustrations are denoted by the symbol ● at the left of the caption.

Summary of Amendments

- References to Model 8 and Model 12 have been added.
- Miscellaneous additions and corrections.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901

Printed in U.S.A.

Preface

This manual presents advanced RPG II programming topics for application programmers and students, who must code programs for IBM System/3:

- Model 6
- Model 8
- Model 10 (Card System)
- Model 10
- Model 12
- Model 15

The System/3 Model 8 is supported by System/3 Model 10 control programming and program products. The facilities described in this publication for the Model 10 are also applicable to the Model 8, although the Model 8 is not referred to. Note that not all devices and features that are available on the Model 10, are available on the Model 8. Therefore, Model 8 users should be familiar with the contents of the *IBM System/3 Model 8 Introduction,* GC21-5144.

PREREQUISITES

This manual assumes that you have coded and tested some basic RPG II programs that include listing records on a printer, simple calculations, group totals, and the use of more than one record type. You may have gained this experience through IBM education courses, programmed instruction courses, or previous data processing experience. *Introduction to RPG II,* GC21-7514, contains some of this basic information.

ORGANIZATION OF THE MANUAL

This publication has eleven chapters. Chapters 1-6 cover information that is basic to most data processing jobs: RPG II program logic, detailed information about writing input, output, and calculation specifications and the concepts and specifications involved in multifile processing. Additional programming topics that you may require for your job are presented in Chapters 7-11: controlling input and output during calculation time, tables, arrays, data structure, and the DEBUG operation.

RELATED PUBLICATIONS

There are numerous IBM System/3 publications containing further information on RPG II. The following are the related reference manuals:

- *IBM System/3 Card System RPG II Reference Manual,* SC21-7500
- *IBM System/3 RPG II Reference Manual,* SC21-7504 (Model 10, Model 12 and Model 15)
- *IBM System/3 Model 6 RPG II Reference Manual,* SC21-7517

If you are programming on a disk system, it would be helpful if you would understand the disk concepts and disk file processing information in the following books before reading this book:

- *IBM System/3 Disk Concepts and Planning Guide,* GC21-7571
- *IBM System/3 RPG II Disk File Processing Programmer's Guide,* GC21-7566

How To Use The Manual

This publication is a *programmer's guide;* it is not intended to serve the same purpose as a reference manual of language specifications and does not replace a reference manual. RPG II programming topics are approached and organized according to their normal use in a data processing job, using examples whenever possible. Unlike a reference manual, individual chapters are self-contained units of information, intended to be read from beginning to end. However, if you desire information about a specific topic, you may go directly to that topic by using the index or the table of contents. If an individual chapter has a special prerequisite topic, that topic is clearly identified on the title page of the chapter.

Although the chapters are complete units, there is a logical progression of topics through chapters 1-6. Therefore, you may wish to read them consecutively. If you have read the *RPG II Programming Fundamentals* Programmed Instruction course, you do not need to read chapters 1-6 consecutively.

For ease of illustration, many of the examples in this book use card-like figures to represent records. This does not imply that a card device must be used for input or output in these situations. Any of several input/output devices might be used, depending on which System/3 model and configuration you are using.

Review Questions

Review questions and answers are provided at the end of each chapter. Where chapters contain several related topics, these questions are grouped by subtopic. If you wish, you may turn to the end of the chapter after you complete each subtopic, to answer the review questions and reinforce what you have learned, before continuing the chapter.

Contents

Chapter 1. RPG II Logic

CHAPTER 1 DESCRIBES:

Basic RPG II logic.

RPG II logic related to indicators.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Basic three-step logic of a data processing job.

Detail time and total time.

Specific steps in a basic RPG II job that includes detail and total operations.

RPG II logic related to the following indicators: 1P, LR, record identifying indicators, field indicators, resulting indicators, halt indicators (H1-H9), overflow indicators (OA-OG, OV), matching records indicator (MR).

Note: You can use the review questions contained in *Review 1* at the end of this chapter to test your comprehension of the chapter. Answers follow the review questions.

INTRODUCTION

What procedures do you follow if you are preparing bills to send to customers? Before you can do anything you need some information. You have to know three things: (1) the customer's balance at the beginning of the month; (2) his purchases; and (3) his payments. Once you have gathered this information, you can perform the necessary calculations to find the amount due. Finally you record this amount on the bill. You go through these same procedures for each customer.

This is a type of job you can easily have your computer do for you. To do the job, however, the computer must know the same things you know. You must, therefore, tell it exactly what information to expect, what to do with the information, and what to give you as a result. This you do through specifications you write: File Description; Extension; Line Counter; Input; Calculation; and Output-Format.

To do this billing job, you do things in a logical order. You read information first. You do calculations second. Finally, as a result of the calculations, you record the amount owed. Then you begin to do the same things 'n the same order for the next customer.

The computer must also do things in a logical order. The information you supply through specifications doesn't give the computer the logic it needs to do your job; the RPG II Compiler supplies this. RPG II logic supplied by the compiler is the framework for your job. When your source program is compiled, your source statements are fitted into the framework of RPG II program logic to make a complete program. The generated program then has all the information it needs to do your job in a logical manner.

What happens if, in doing your billing job, you find that a customer paid more than he owed? You know immediately that he has a credit balance and indicate so on the invoice. How does the RPG II program recognize this situation? And how does it know what to do when such a situation occurs?

The RPG II program uses signals which tell it when a particular situation occurs and what to do when that situation does occur. These signals are known as indicators. There are many different kinds of indicators which signal many different situations. You, as a programmer, must know how to specify the indicators so that they signal to the computer what you want them to.

RPG II logic is built around these indicators. Their status (on or off) affects the sequence of the program's operations. The logic is set up to test the status of various indicators at specific times. By testing indicators, the program knows what to do next.

RPG II program logic is designed to take care of all types of jobs. You must understand this logic to write specifications which make correct use of it.

Because the logic is a rather complex topic, it is described segment by segment. However, when you have finished reading this section, you will have a picture of how the complete RPG II program logic works.

BASIC DATA PROCESSING LOGIC

Usually, all records in a file of input records are not read at once. Your computer probably is not large enough to store and work with information from all records at the same time. Therefore, records are read one at a time. Three steps, as shown in Figure 1-1, are done for each record read.

The phrase *program cycle* refers to all the operations performed from the time one record is read until the next record is read. One program cycle is therefore one revolution around the circle used to illustrate the program logic. Since one program cycle (one revolution) is needed for each record read, many program cycles are required for every job.

Consider how the three step logic shown in Figure 1-1 works for a job which requires a detailed listing of purchases made by each customer. The input file is in ascending order by customer number. Each record contains customer name (NAME), number (NUM), and charge (CHRG). Information from the record is merely transferred to the printed page. One line is printed for each record read. Each record read is known as a detail record and each line printed is a detail line.

The job begins: the first record is read. No calculations are performed. A record is then printed. This ends the first program cycle. The second begins with the reading of another record. Figure 1-2 shows the input and output of the detail printing job.

Suppose, however, instead of merely listing the charges made by each customer you also wish to find the total charges for each customer, as shown in Figure 1-3.

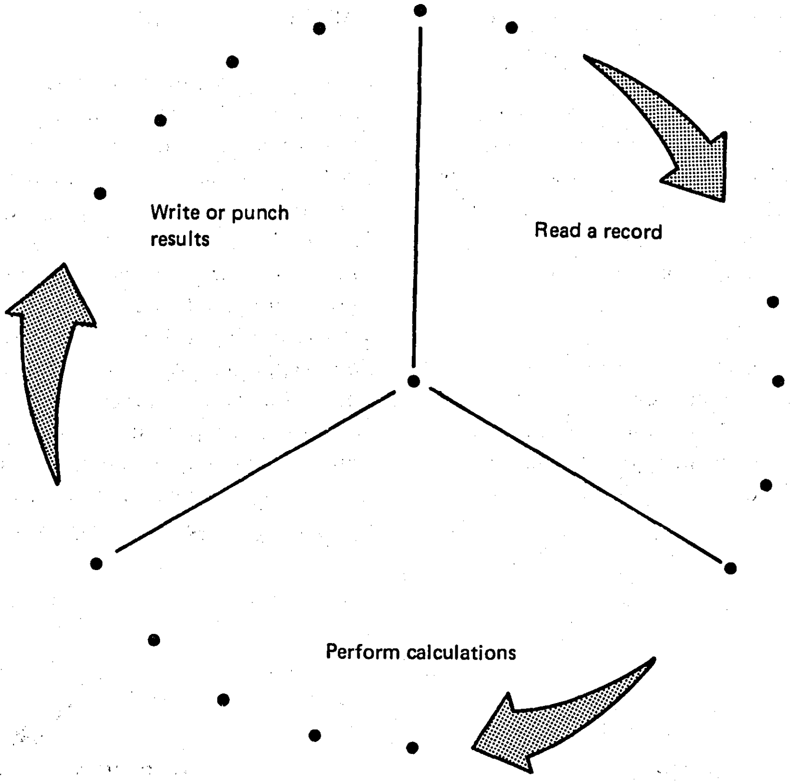
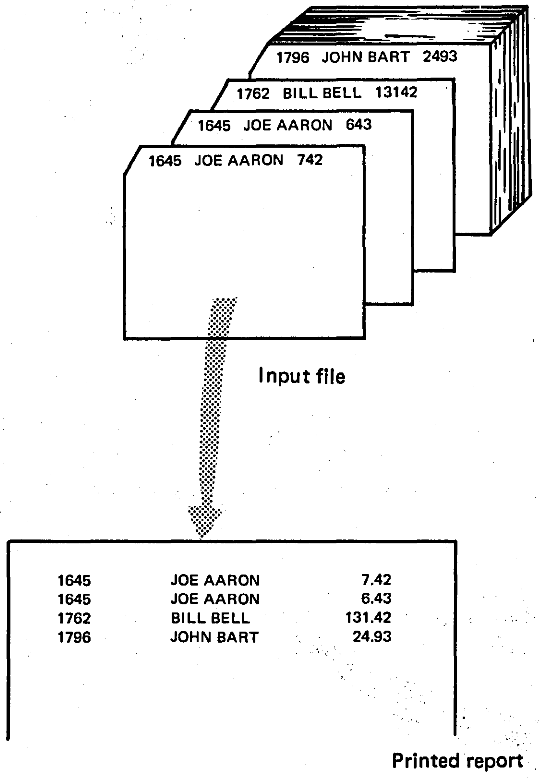
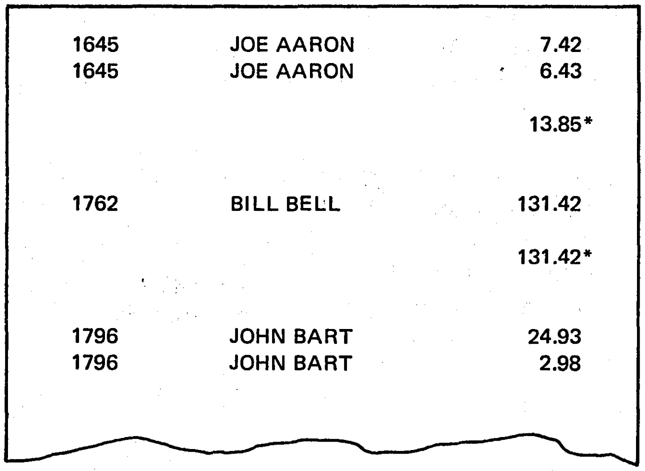


Figure 1-1. Basic Logic of a Data Processing Job



```
1645    JOE AARON      7.42
1645    JOE AARON      6.43
1762    BILL BELL    131.42
1796    JOHN BART     24.93
```

Figure 1-2. Detail Printing Job



```
1645          JOE AARON           7.42
1645          JOE AARON           6.43

                                 13.85*

1762          BILL BELL         131.42

                                131.42*

1796          JOHN BART          24.93
1796          JOHN BART           2.98
```

Figure 1-3. Calculating and Printing Totals

To do this, you must do calculations to accumulate a total in addition to printing out individual (detail) records. But when do you print out the total you have calculated? The total for a customer, of course, should be printed after all detail records for that customer have been printed (Figure 1-3). However, in the three-step logic discussed so far, there is no provision for printing a total record. Neither is there a way to distinguish between individual input records in order to determine when all records for a customer have been read.

If the RPG II program used only the three-step logic, it would not be able to do this job and many others like it. It could adequately work with information from only one record at a time, as in the detail printing job. It could not correctly do operations to accumulate data from several records.

BASIC RPG II LOGIC

RPG II logic, therefore, is an extended version of this 3-step logic. It calls for calculations and output operations to be done at two different times in one program cycle (see Figure 1-4). The names *detail* and *total* have been given to the times at which calculation and output operations are performed. Total time, as the name suggests, is the time in which total operations are done on data accumulated from a group of related records. The printing of total charges for Joe Aaron (Figure 1-3) is an example of a total time operation. Detail time is the time in which operations are performed for individual records. An example of a detail time operation is the printing of an individual charge for Joe Aaron. Remember, detail operations are done for every record read, but total operations are done only after a certain group of records are read (see Figure 1-5).

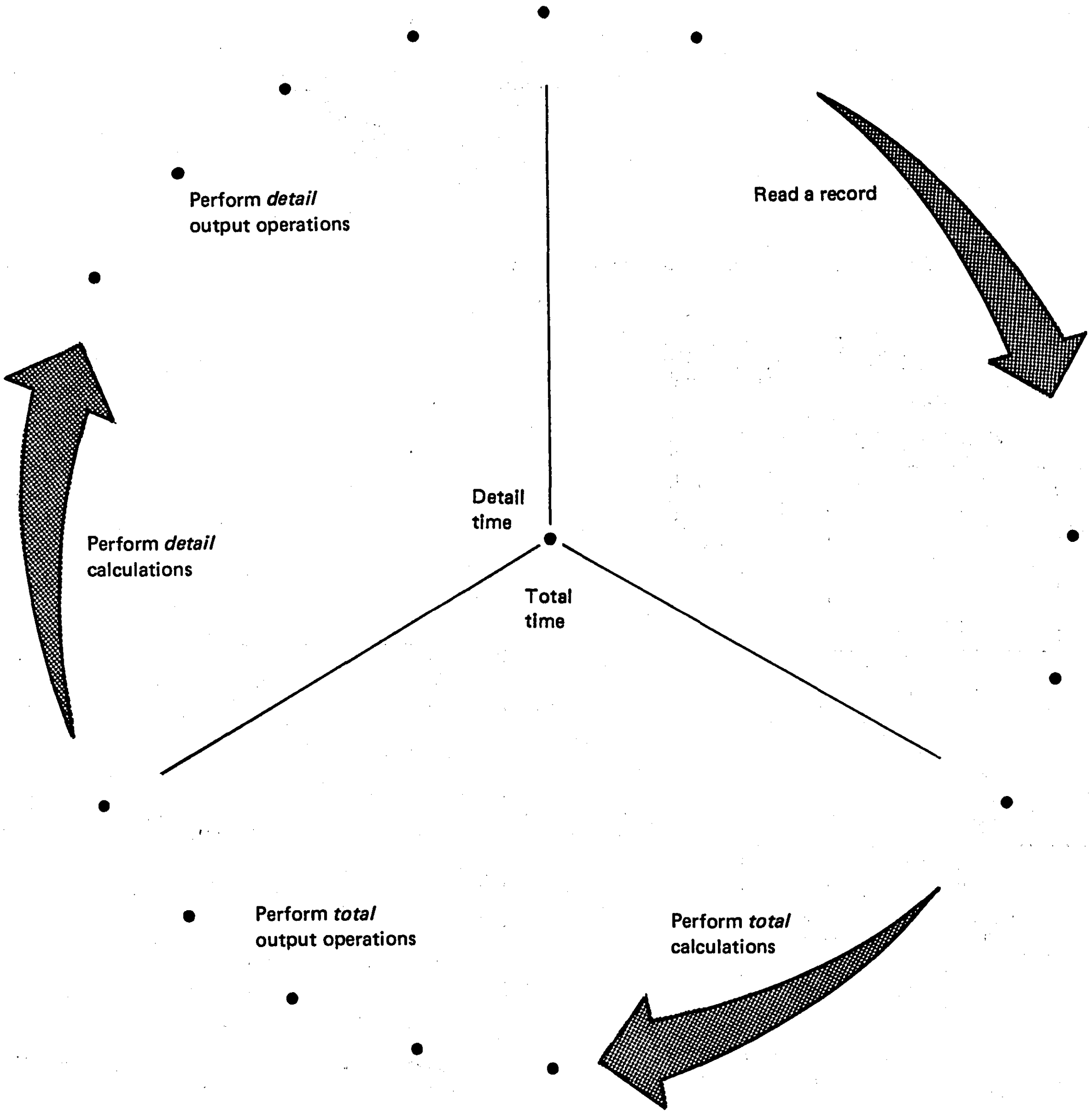

Read a record
Perform *total* calculations
Perform *total* output operations
Perform *detail* calculations
Perform *detail* output operations
Detail time
Total time

Figure 1-4. Basic RPG II Logic

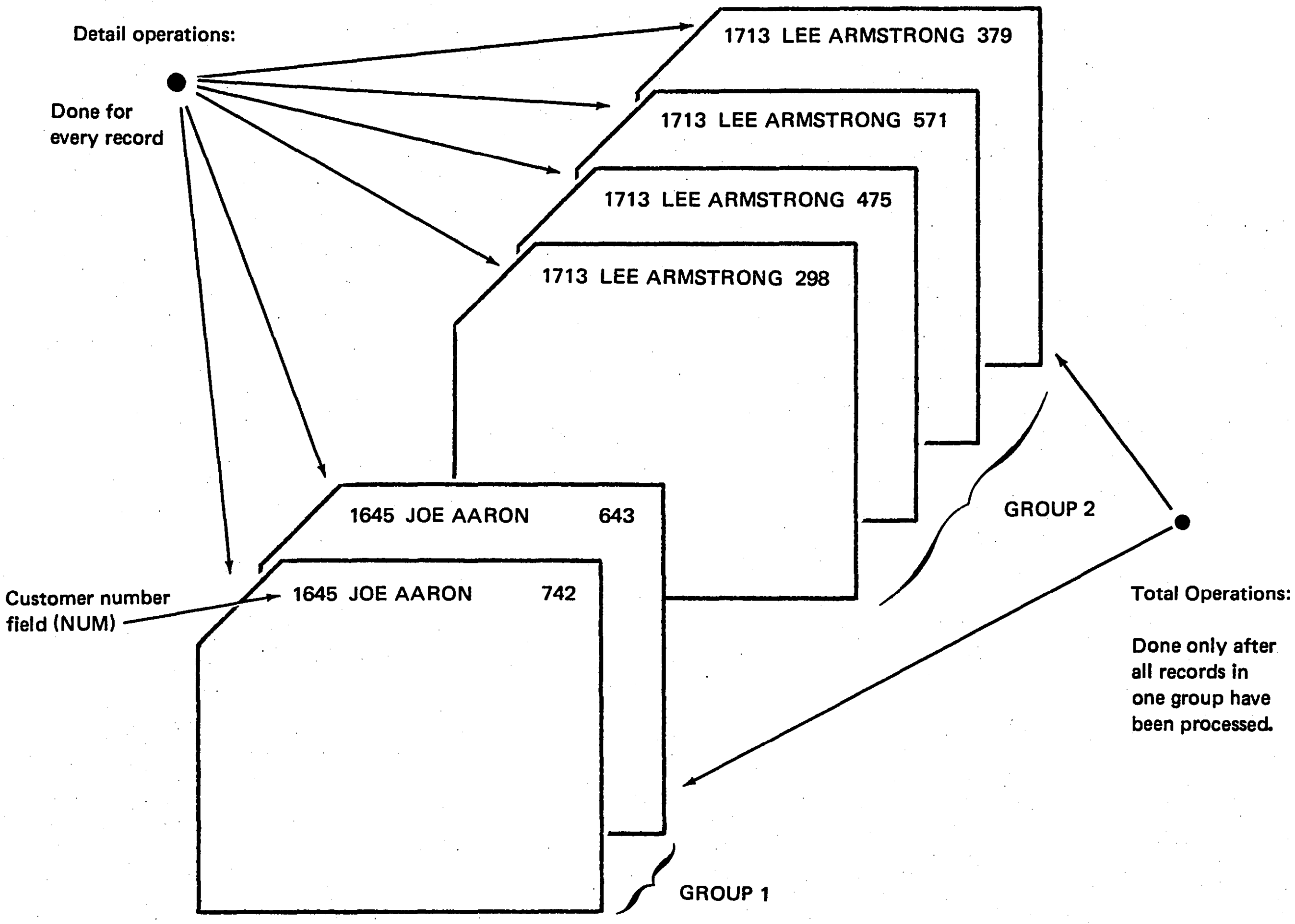


Figure 1-5. Detail Versus Total Operations

Because this basic RPG II logic is only a framework for your job, you have to supply additional information so that your job will be complete. Only then will your program work correctly. For example, the RPG II compiler supplies your program with the logic framework which enables it to do detail and total operations. But you must tell it *when* total operations should be done and which calculation and output operations are to be done at detail time and which are to be done at total time.

Remember that the only way you can tell the program what to do in certain situations is to use indicators. Control level indicators are used to tell the program:

1. When to do total operations.

2. What operations are total operations.

If you were finding total monthly charges for each customer, how would you know when to record totals for each customer? When you encounter a record with a different customer number in the NUM field, you know that you have gathered all the information for one customer. (You could use the NAME field to tell you this, but there is a chance that two customers may have the same name.) You would then record that total before gathering information for the next customer.

Any field used to control and direct processing is known as a *control field.* You indicate to the compiler program which field is a control field by assigning one of the control level indicators (L1-L9) to the field in columns 59-60 of the Input sheet. You also use this same control level indicator to tell the program which calculations are total calculations by entering the indicator in columns 7-8 of the Calculation sheet. Those calculations that are not conditioned by a control level indicator (in columns 7-8) are detail calculations. Control level indicators are not used in the Output-Format sheet to indicate detail and total records. Rather a T is used in column 15 to indicate a total output operation; and H or D is used to indicate an operation done at detail time.

Specific Steps in Basic RPG II Logic

Figure 1-4 shows very generally the sequence of events in an RPG II program cycle. The RPG II logic actually consists of definite steps taken during the cycle. When you do a job you mentally ask yourself questions such as, "Do I do this now? Do I have all my information? What should I do next?" RPG II logic also asks questions. It uses your program to find the answer and thus determines what to do next. The questions, and specific steps taken based on the answers to the questions, are shown in Figure 1-6.

According to RPG II logic, after a record is read the program checks to see if information in the control field of the record just selected is different from the control field information in the previous record. (The program always saves the control field information so that it can make a comparison.) If there is a change, the proper control level indicator (the one you assigned) is turned on. This means that all records from one group have been read. All total operations can then be performed. Control level indicators are always turned off before the next record is read.

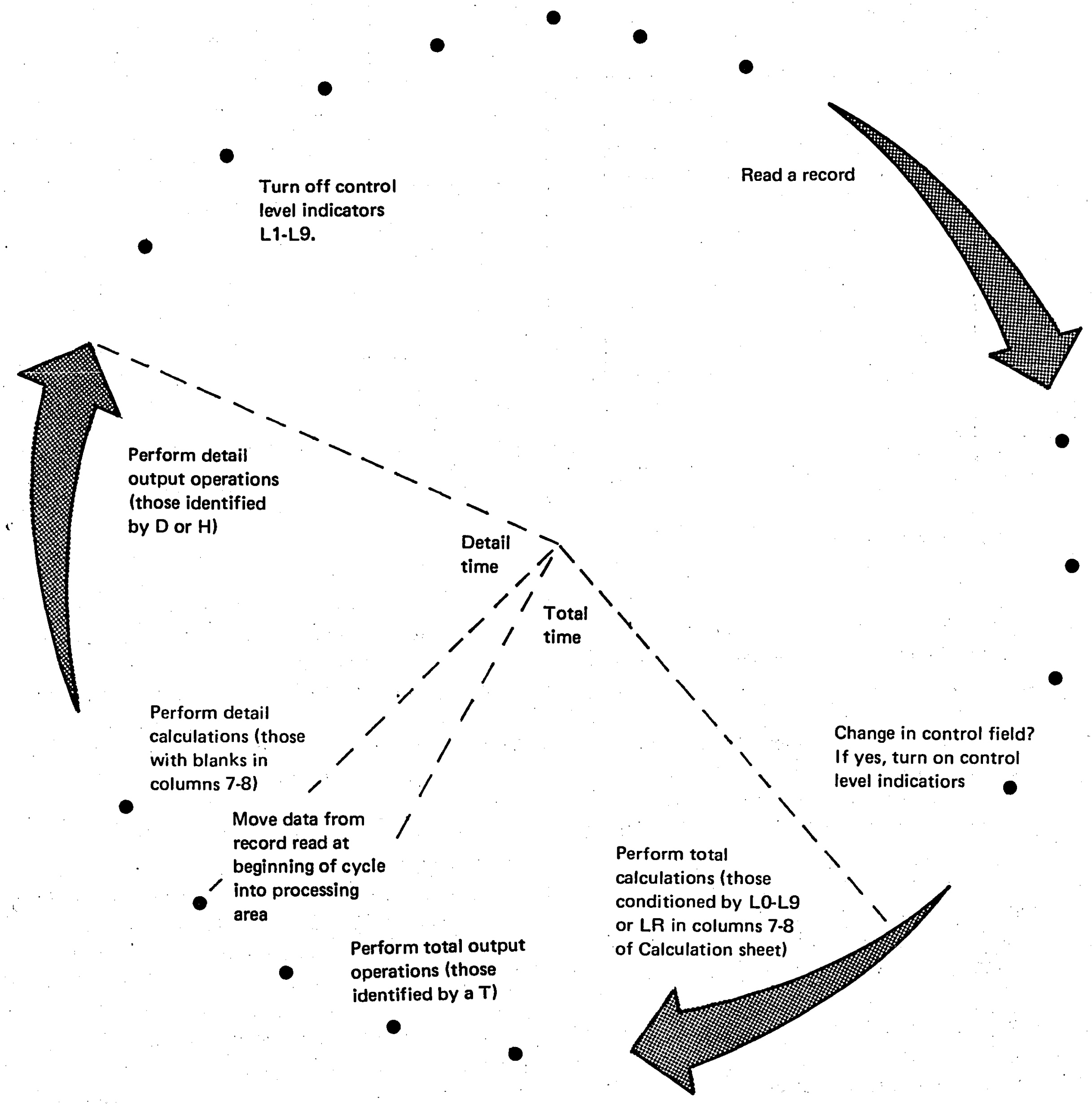

Read a record
Change in control field? If yes, turn on control level indicatiors
Perform total calculations (those conditioned by L0-L9 or LR in columns 7-8 of Calculation sheet)
Perform total output operations (those identified by a T)
Move data from record read at beginning of cycle into processing area
Perform detail calculations (those with blanks in columns 7-8)
Detail time
Total time
Perform detail output operations (those identified by D or H)
Turn off control level indicators L1-L9.

Figure 1-6. Steps in RPG II Total and Detail-Time Logic

Notice the step between total and detail time. Here data from the record read at the beginning of the cycle is moved into a processing area and becomes available to use in calculations and output. Data from this record is not available at total time. Total operations are performed only on data accumulated from previous records. Detail operations on the record which caused the control level indicator to be turned on are done only after total operations for previous records.

Why are total operations done before detail operations? Think of what would happen if the record which caused the control level indicators to be turned on were processed. Information on this record would be added to information from records in the previous group. As a result, the totals printed would be in error since they contained information from one record in the next group (the record just read). To prevent data from the first record in a new control gropp from being accumulated in the totals for the previous group, total operations are done before detail operations.

First Program Cycle

When control fields are specified for a record, the first program cycle may be slightly different from the others.

Control level indicators are turned on by the first record containing control fields. This happens because contents of the control fields on this record are different from the blank control field areas that were in main storage before the record was read. To prevent printing of blank totals on the first cycle, RPG II logic causes total operations to be bypassed on the first cycle.

Note: If the initial input records do not contain a control field, total calculations and total output operations are bypassed on each program cycle through and including the first cycle in which a record with control fields is read.

Summary of Basic RPG II Logic

Figure 1-7 shows specifications for the group printing job previously discussed and the logic for the first four program cycles. Follow the logic involved in each program cycle step by step. Remember, the cycle repeats itself from the time the program is started until the last record is processed.

Be sure you understand this basic logic before proceeding further.

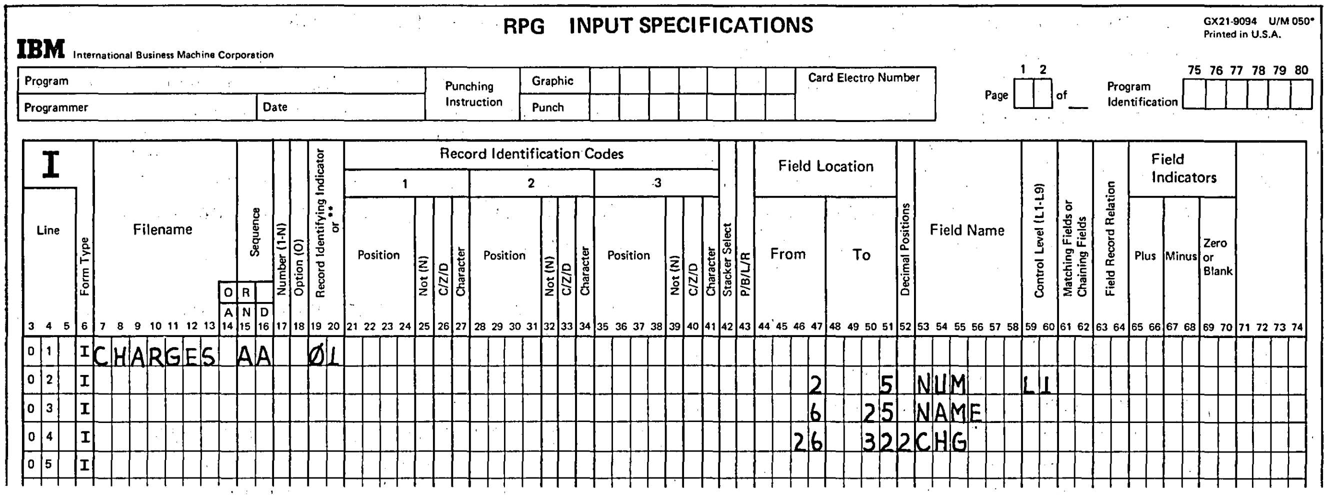
RPG II LOGIC RELATED TO INDICATORS

It was previously stated that RPG II logic is built around indicators. This section discusses how logic related to indicators fits into the basic RPG II detail and total time logic shown in Figure 1-4.

In your specifications, you use indicators to tell the program what to do and when to do it. Although you use indicators, you do not set them. Naturally the indicators don't turn on and off by themselves. The compiler supplies logic which is needed to control the setting of indicators.

Indicators are set to signal various conditions that occur during the execution of a program. In addition to setting indicators, RPG II logic also causes tests to be made for various indicators at certain times in the program cycle. Specific operations are performed as a result of these tests.

It is very easy to think that an indicator is on when it really is off or vice versa. It is extremely important that you know when indicators are on and when they are off in the program cycle. Many programs fail just because the programmer did not understand RPG II logic concerning indicators. The following paragraphs will discuss the time in the program cycle at which indicators are set and the time at which they are tested.



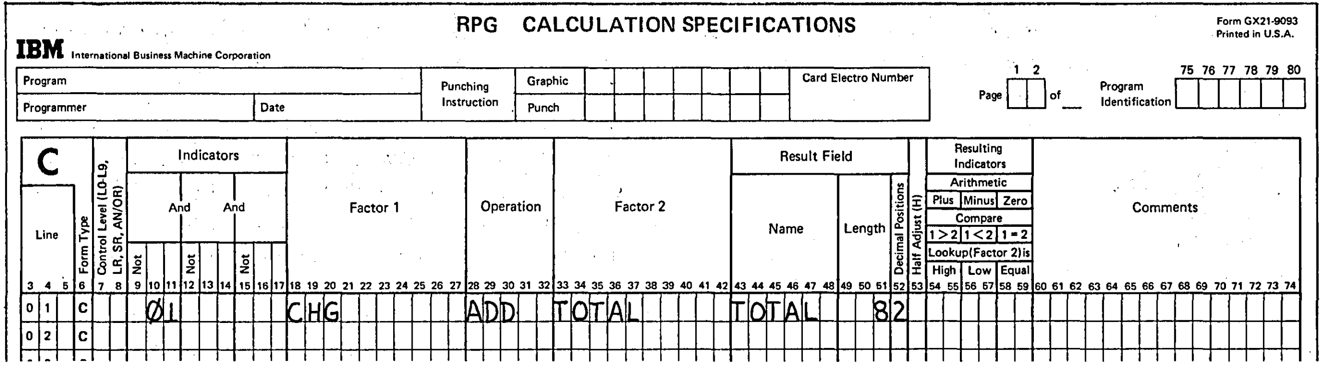
IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program / Programmer / Date / Punching Instruction: Graphic, Punch / Card Electro Number / Page 1 2 of / Program Identification 75 76 77 78 79 80

```
  01 ICHARGES AA  Ø1
  02 I                                        2   5 NUM   L1
  03 I                                        6  25 NAME
  04 I                                       26  322CHG
  05 I
```



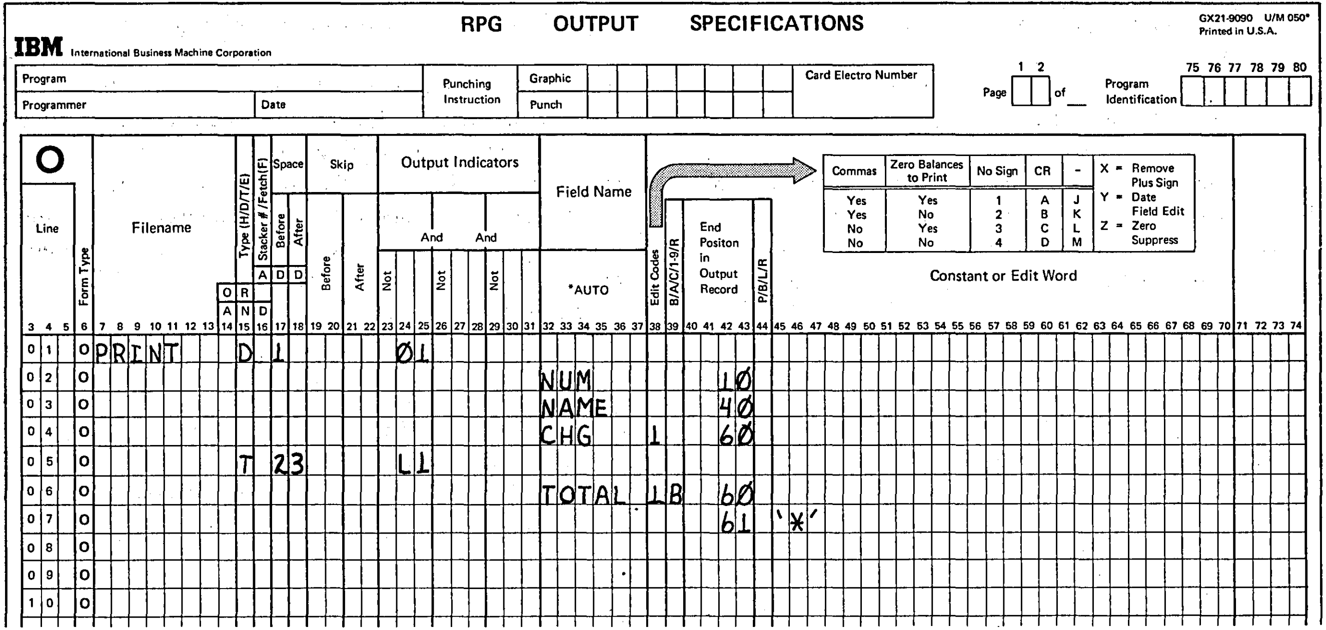
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Program / Programmer / Date / Punching Instruction: Graphic, Punch / Card Electro Number / Page 1 2 of / Program Identification 75 76 77 78 79 80

```
  01 C   Ø1      CHG       ADD  TOTAL     TOTAL   82
  02 C
```



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program / Programmer / Date / Punching Instruction: Graphic, Punch / Card Electro Number / Page 1 2 of / Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

```
  01 OPRINT   D 1      Ø1
  02 O                         NUM       1Ø
  03 O                         NAME      4Ø
  04 O                         CHG   1   6Ø
  05 O        T 23     L1
  06 O                         TOTAL 1B  6Ø
  07 O                                   61 '*'
  08 O
  09 O
  10 O
```

Figure 1-7 (Part 1 of 5). Illustration of Detail and Total Time

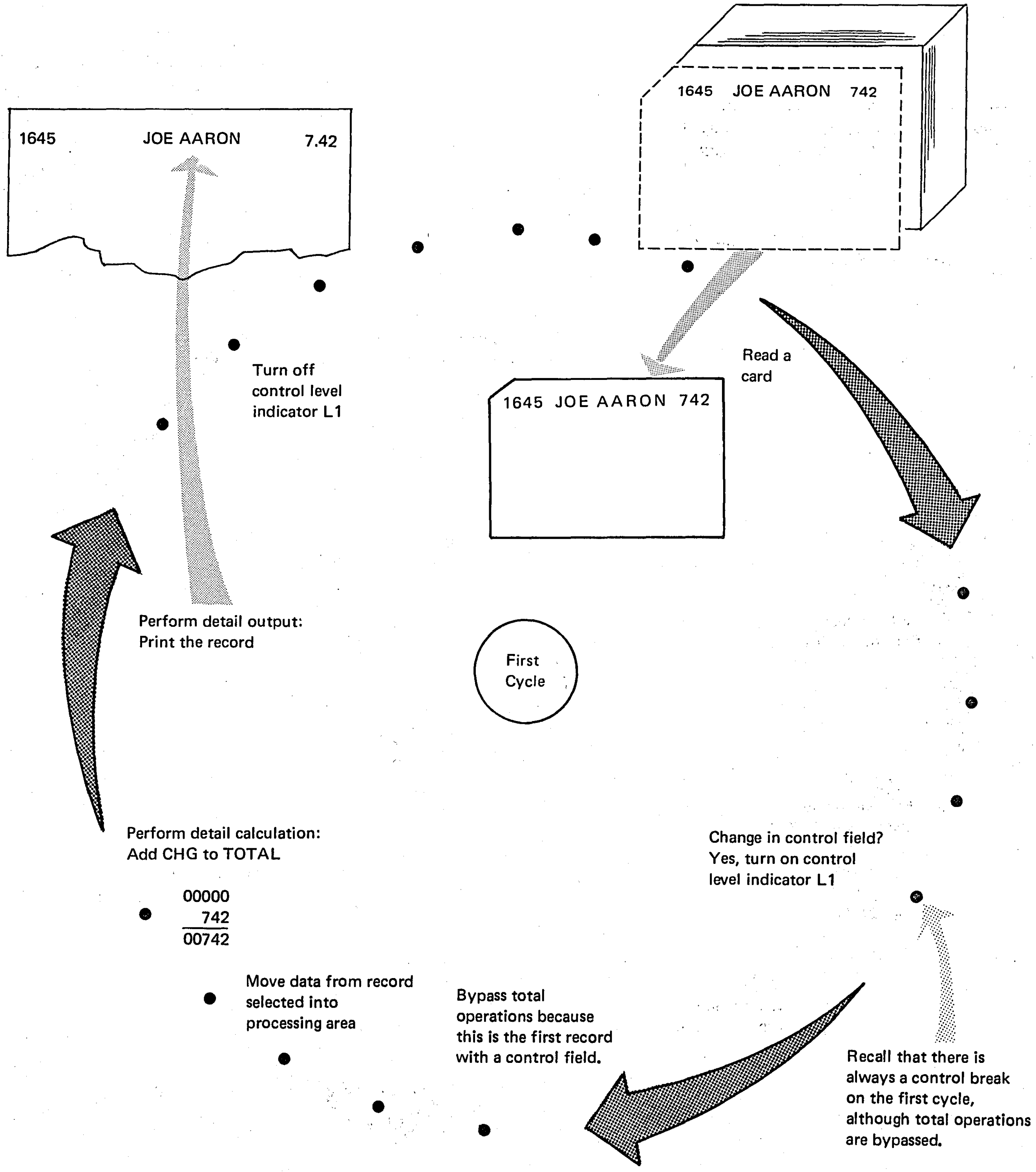

1645 JOE AARON 7.42
1645 JOE AARON 742
Read a card
1645 JOE AARON 742
Change in control field? Yes, turn on control level indicator L1
Recall that there is always a control break on the first cycle, although total operations are bypassed.
Bypass total operations because this is the first record with a control field.
Move data from record selected into processing area
Perform detail calculation: Add CHG to TOTAL
00000
742
00742
Perform detail output: Print the record
Turn off control level indicator L1
First Cycle

Figure 1-7 (Part 2 of 5). Illustration of Detail and Total Time

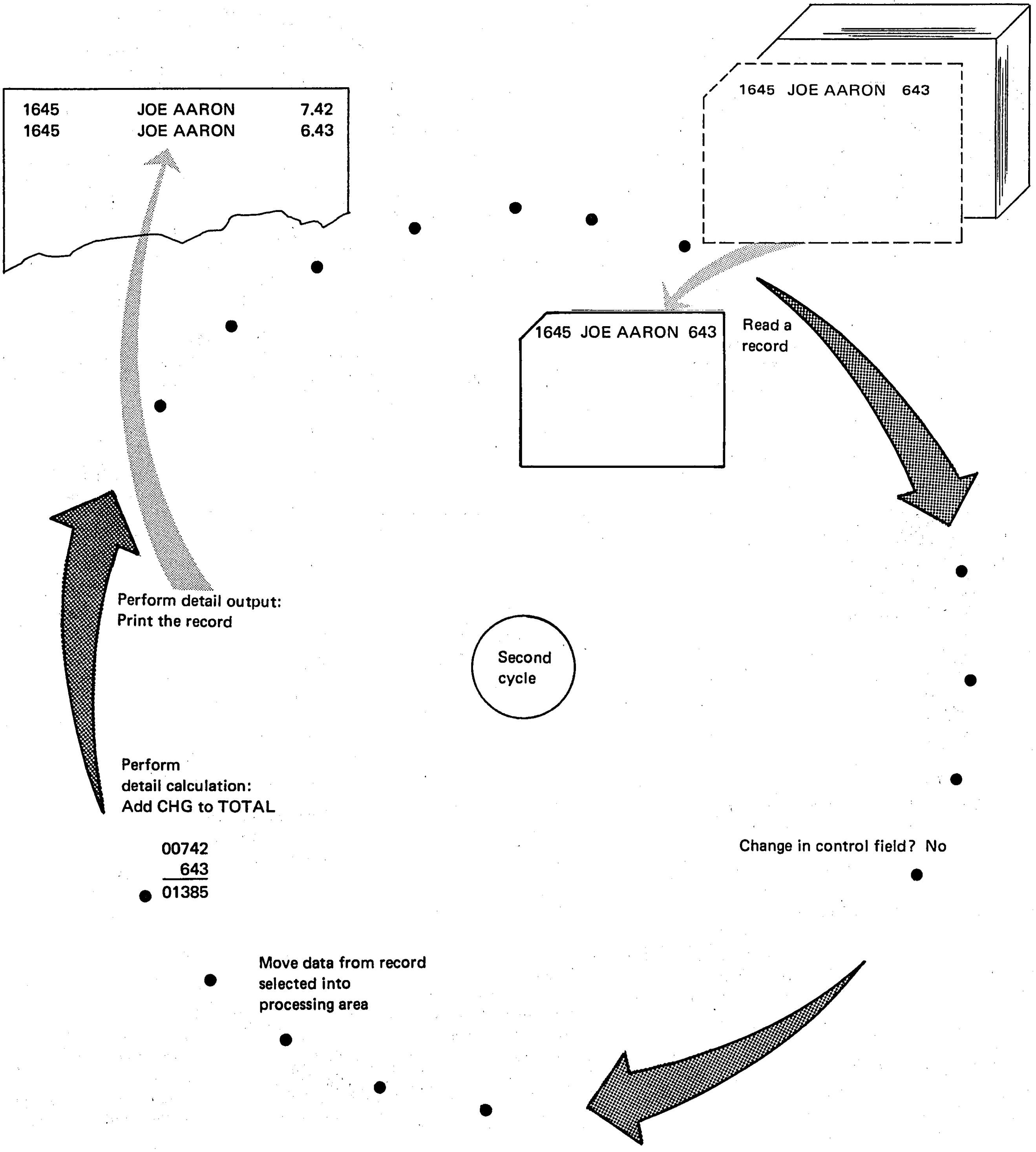

1645 JOE AARON 7.42
1645 JOE AARON 6.43
1645 JOE AARON 643
1645 JOE AARON 643
Read a record
Second cycle
Change in control field? No
Move data from record selected into processing area
Perform detail calculation: Add CHG to TOTAL
00742
643
01385
Perform detail output: Print the record

Figure 1-7 (Part 3 of 5). Illustration of Detail and Total Time

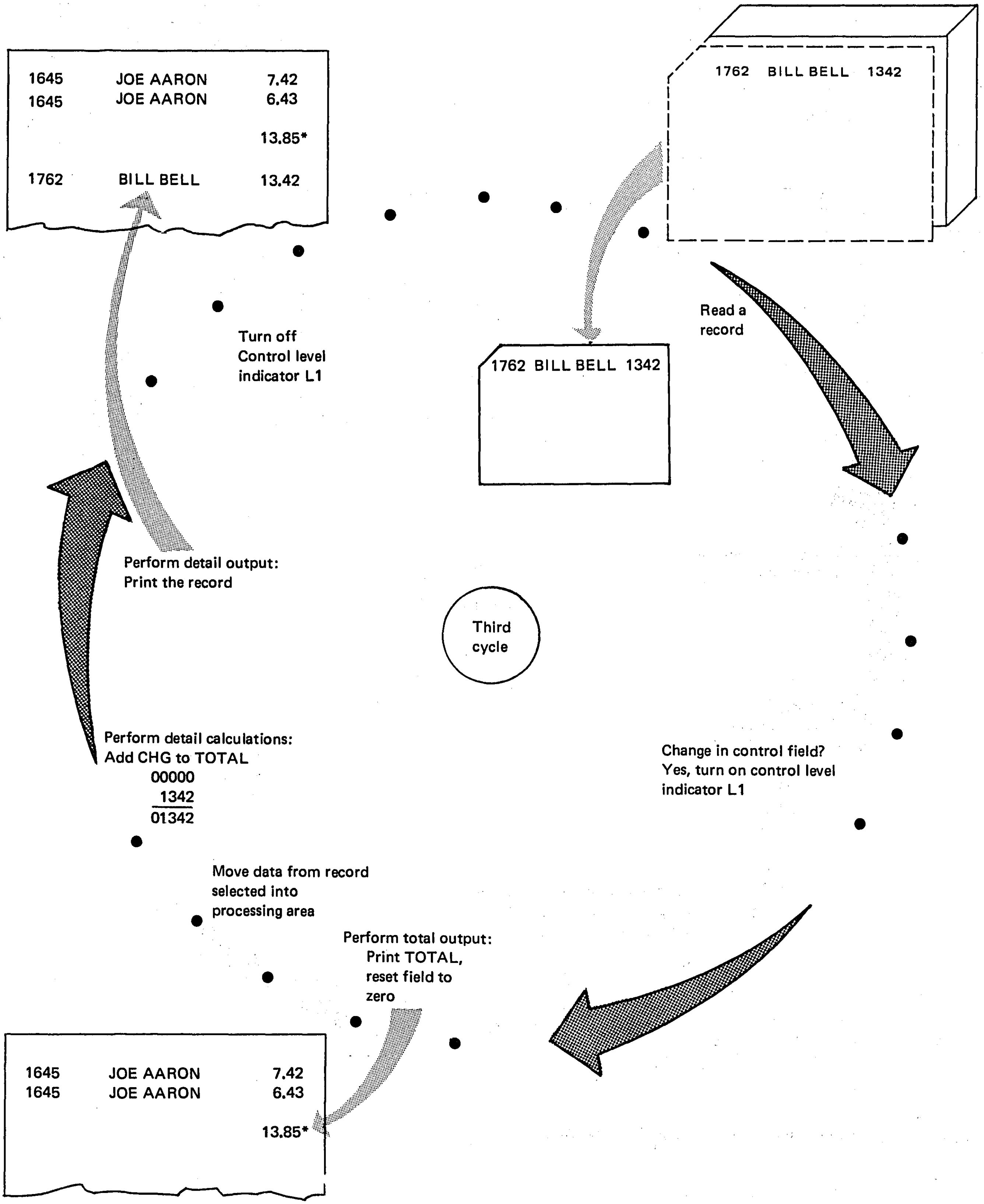

1645 JOE AARON 7.42
1645 JOE AARON 6.43
13.85*
1762 BILL BELL 13.42
1762 BILL BELL 1342
Read a record
1762 BILL BELL 1342
Turn off Control level indicator L1
Perform detail output: Print the record
Third cycle
Change in control field? Yes, turn on control level indicator L1
Perform detail calculations: Add CHG to TOTAL
00000
1342
01342
Move data from record selected into processing area
Perform total output: Print TOTAL, reset field to zero
1645 JOE AARON 7.42
1645 JOE AARON 6.43
13.85*

Figure 1-7 (Part 4 of 5). Illustration of Detail and Total Time

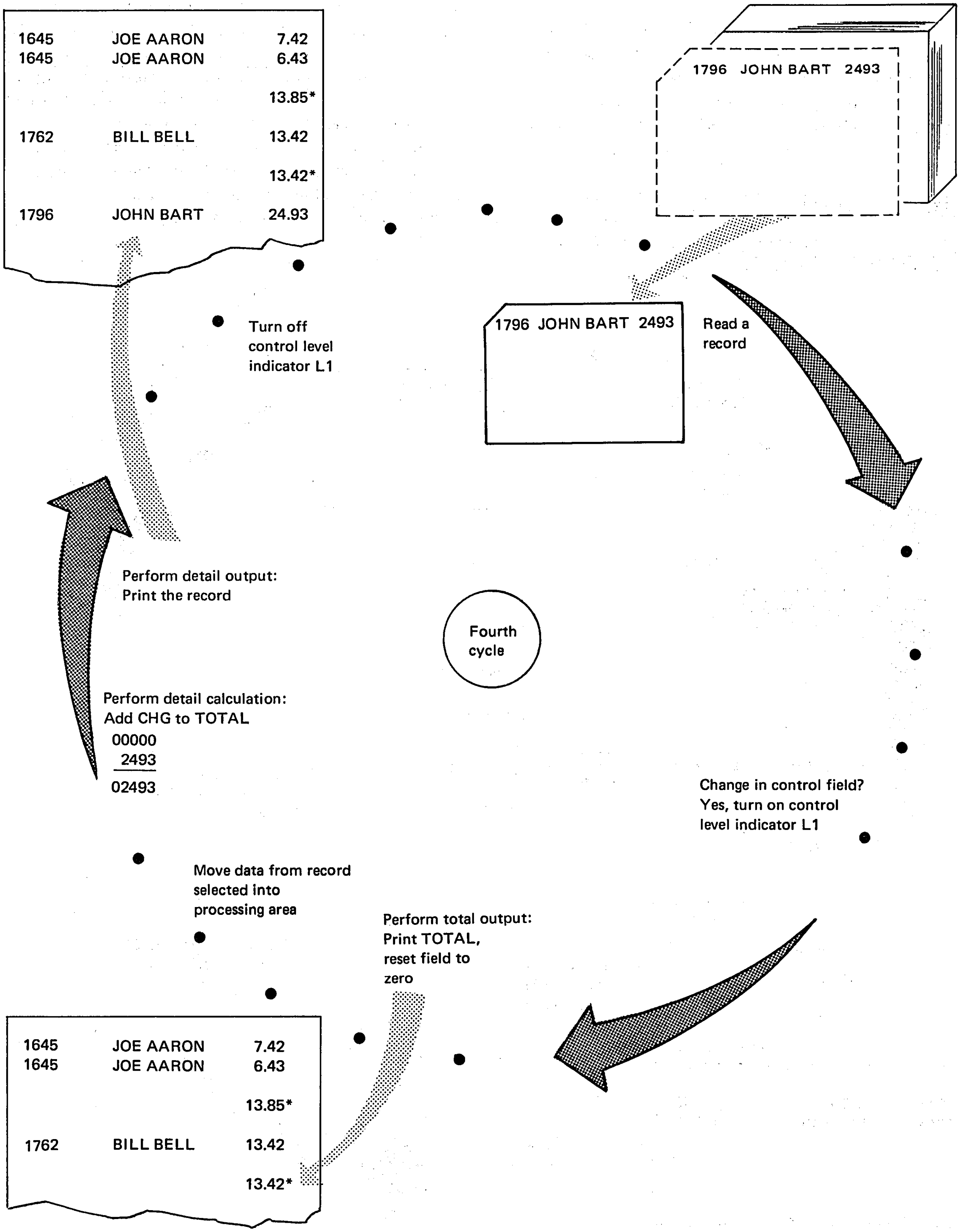

1645 JOE AARON 7.42
1645 JOE AARON 6.43
13.85*
1762 BILL BELL 13.42
13.42*
1796 JOHN BART 24.93
1796 JOHN BART 2493
1796 JOHN BART 2493
Read a record
Turn off control level indicator L1
Perform detail output: Print the record
Fourth cycle
Perform detail calculation: Add CHG to TOTAL
00000
2493
02493
Change in control field? Yes, turn on control level indicator L1
Move data from record selected into processing area
Perform total output: Print TOTAL, reset field to zero
1645 JOE AARON 7.42
1645 JOE AARON 6.43
13.85*
1762 BILL BELL 13.42
13.42*

Figure 1-7 (Part 5 of 5). Illustration of Detail and Total Time

1P (First Page) Indicators

It was stated before that the first program cycle is slightly different from the others because total operations are bypassed on the first cycle. Another difference in the first cycle is that the first page indicator (1P) is on during the beginning of the cycle. Any records conditioned by the 1P indicator are printed *before* the first record is read.

This indicator is used to condition records which are to be printed on the first page of a report. These records are usually headings used to identify information found on the page, but may also be detail lines.

The 1P indicator is turned on only for the beginning of the first cycle. It is turned off before a record is read and is never used again during the program (see Figure 1-8).

Notice in Figure 1-8 that the program performs 1P output and other heading and detail output *first* when it is started. *This is always true.* In any program, 1P output and any other heading or detail output for which specified conditions have been met is performed before the first record is read. On succeeding cycles, however, it is usually easier to think of reading a record as the first step in the cycle.

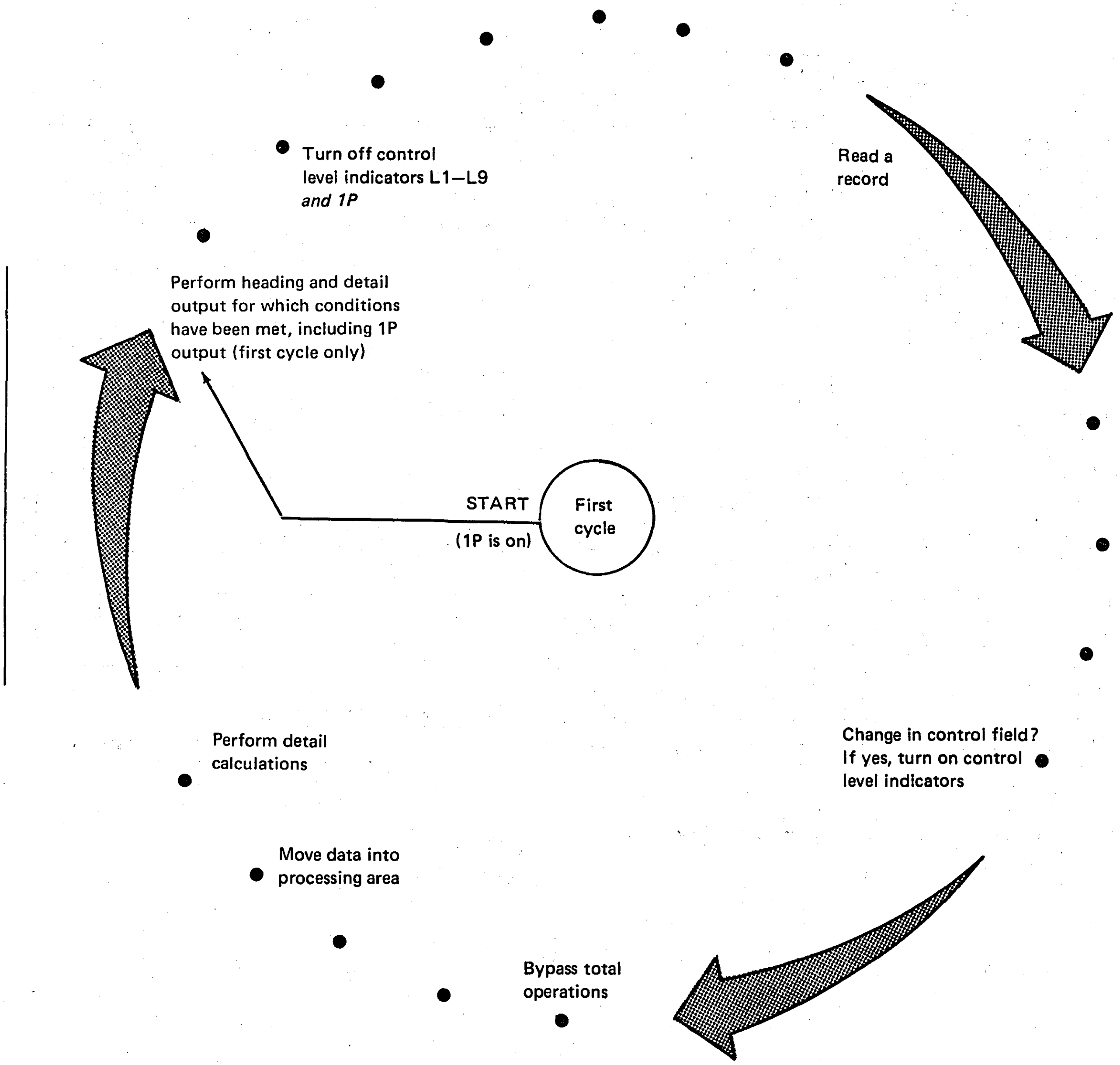
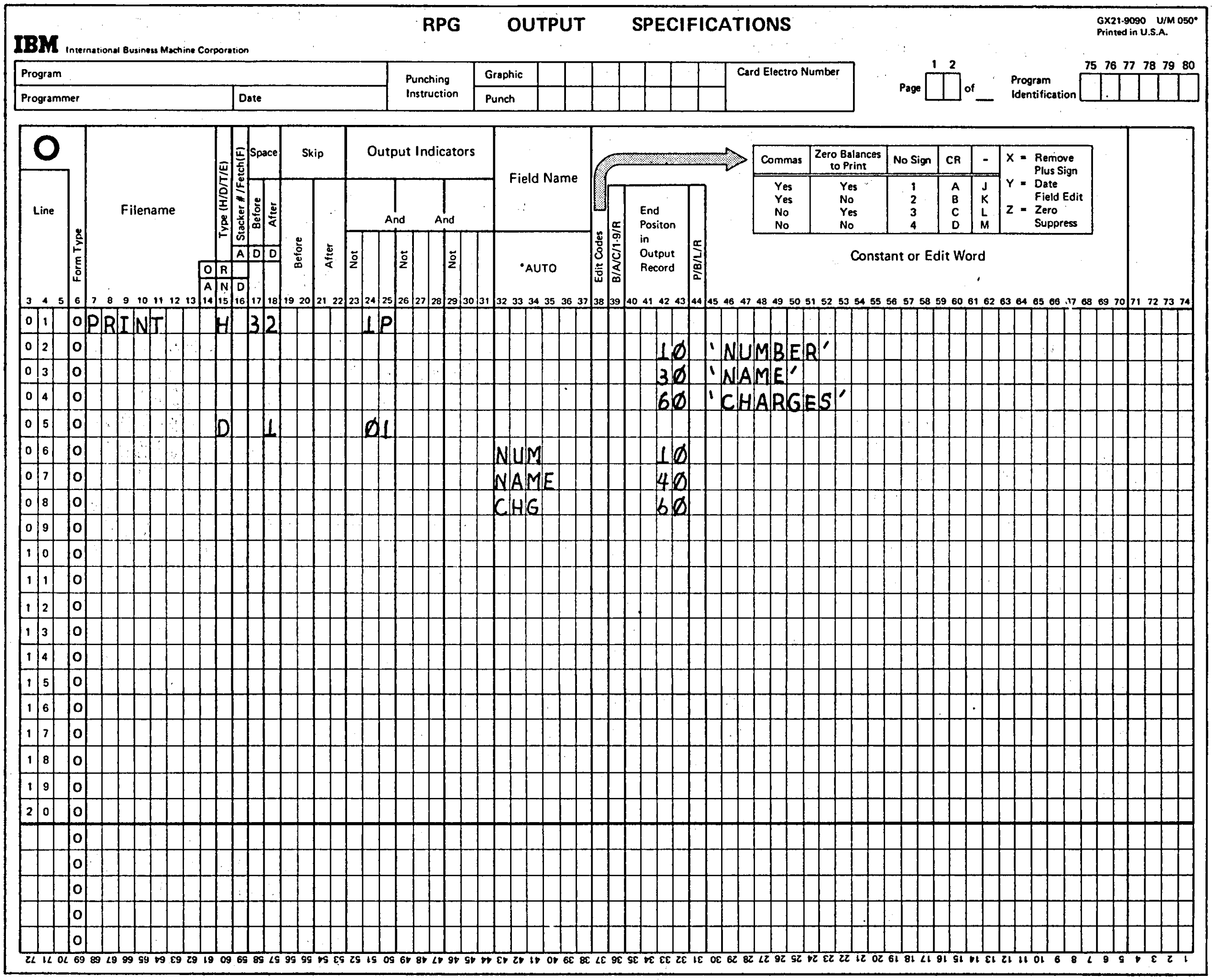

Turn off control level indicators L1—L9 *and 1P*
Read a record
Perform heading and detail output for which conditions have been met, including 1P output (first cycle only)
START
(1P is on)
First cycle
Change in control field? If yes, turn on control level indicators
Perform detail calculations
Move data into processing area
Bypass total operations

Figure 1-8. Logic for the First Page (1P) Indicator

Assume that a heading is desired on the report created by the previous example (Figure 1-7). The heading should be printed on the first page before any records are printed. Thus the heading line is conditioned by 1P (see Figure 1-9). Figure 1-10 shows what happens in the first cycle according to RPG II logic.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page □□ of ___
Program Identification

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	H	3	2	1P			
02	O							1Ø	'NUMBER'
03	O							3Ø	'NAME'
04	O							6Ø	'CHARGES'
05	O		D		1	Ø1			
06	O						NUM	1Ø	
07	O						NAME	4Ø	
08	O						CHG	6Ø	
09	O								
10	O								
11	O								
12	O								
13	O								
14	O								
15	O								
16	O								
17	O								
18	O								
19	O								
20	O								

Figure 1-9. Heading Line Conditioned by the First Page Indicator

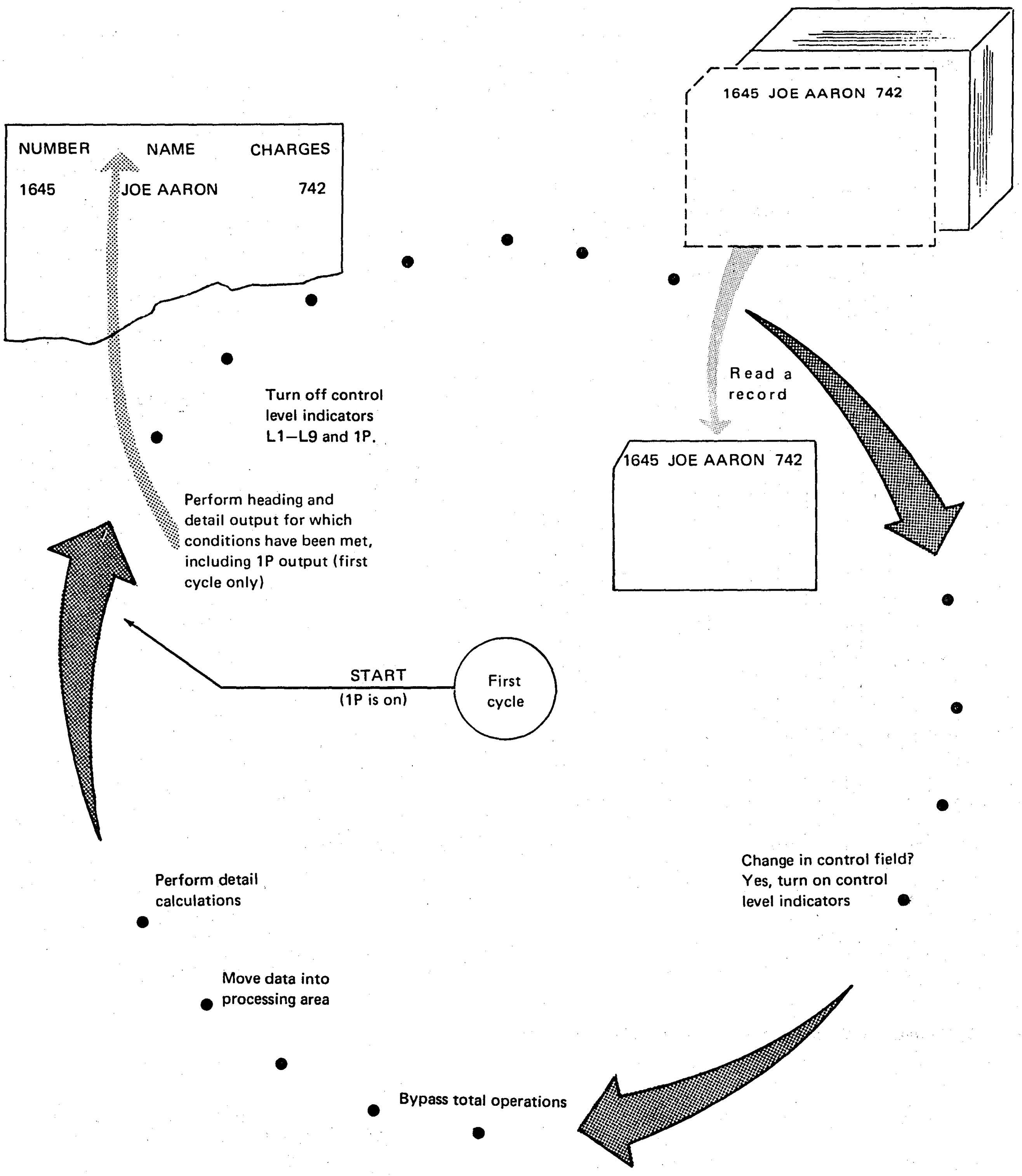

1645 JOE AARON 742
NUMBER
NAME
CHARGES
1645
JOE AARON
742
Read a record
1645 JOE AARON 742
Turn off control level indicators L1–L9 and 1P.
Perform heading and detail output for which conditions have been met, including 1P output (first cycle only)
START
(1P is on)
First cycle
Change in control field? Yes, turn on control level indicators
Perform detail calculations
Move data into processing area
Bypass total operations

Figure 1-10. Program Cycle Illustrating the 1P Indicator

Last Record Indicator (LR)

The last program cycle is also a little different from the others. When the record with a /* in positions 1 and 2 is read, the LR (last record) indicator is turned on. Since the /* record has no data on it, detail operations need not be performed. Thus RPG II logic is set up so that detail operations are not done when LR is on (see *Note*). Total operations are done. The program then ends.

When the last record indicator is turned on, all control level indicators are also turned on. Thus all total operations conditioned by L1-L9 and LR are performed. See Figure 1-11 for specific steps in the end of job logic.

You use LR to condition all operations done at the end of the job. These usually include the calculating of totals for all records and/or writing or punching summary information. Suppose the previous example (Figure 1-7), which found total charges for each customer, required the statement *List Complete as of* ___________ (date job was run). Since this is to print out after all records have been processed, it is conditioned by LR (see Figure 1-12). Figure 1-13 shows what happens during the last program cycle according to RPG II logic.

Note: Detail operations are done if LR has been turned on during calculations, rather than by reading a /* record. However, when LR is turned on in calculations, the other control level indicators are not turned on.

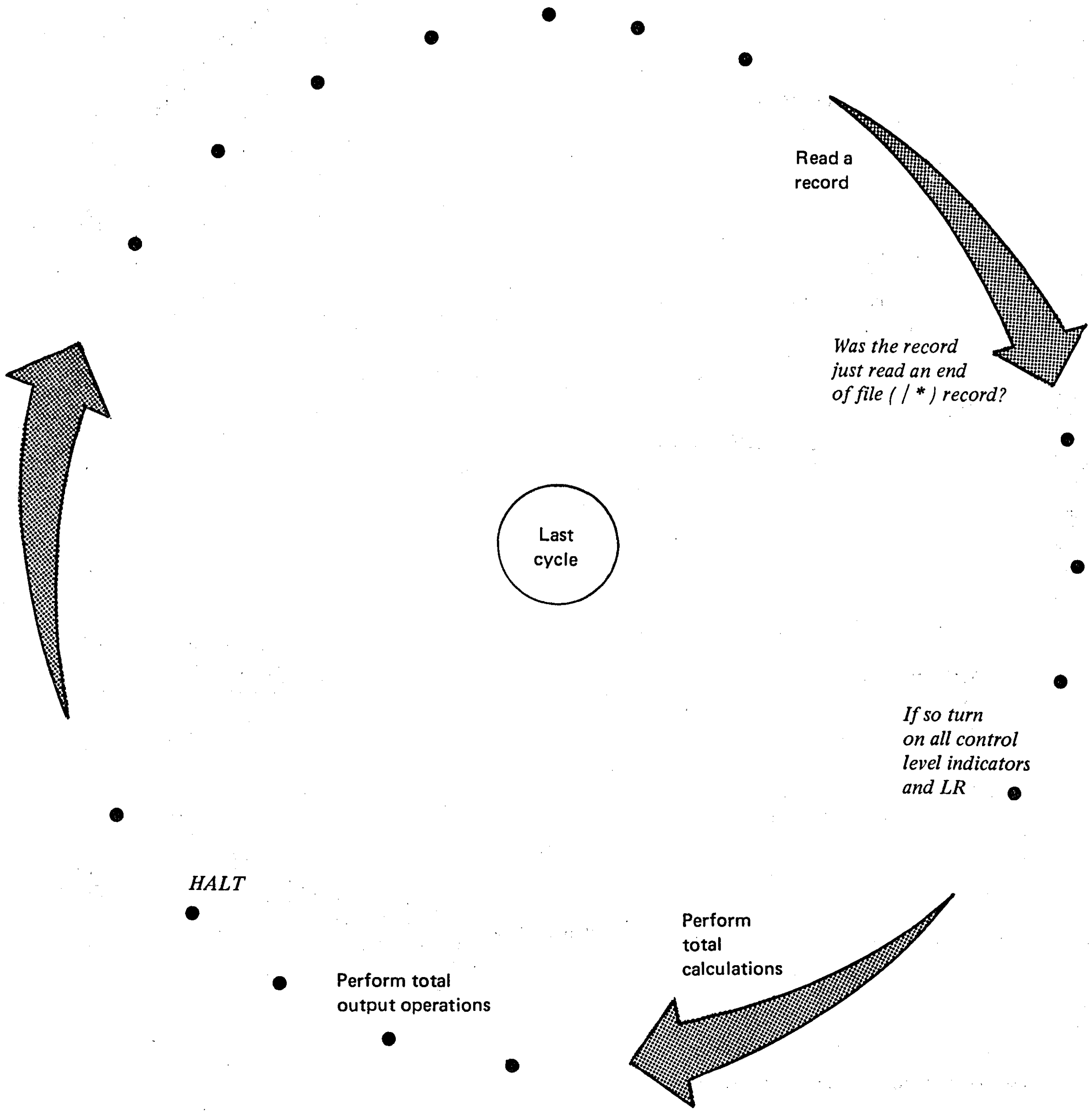
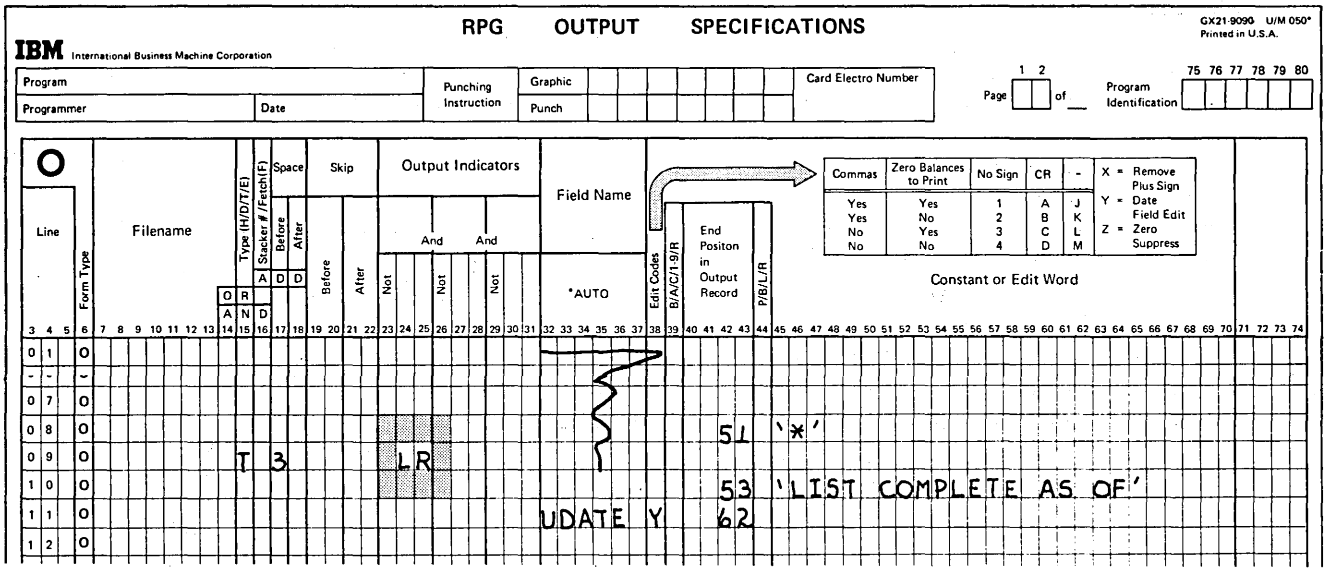

Read a record
*Was the record just read an end of file (/ *) record?*
Last cycle
If so turn on all control level indicators and LR
Perform total calculations
Perform total output operations
HALT

Figure 1-11. Logic for the Last Record (LR) Irfdicator



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

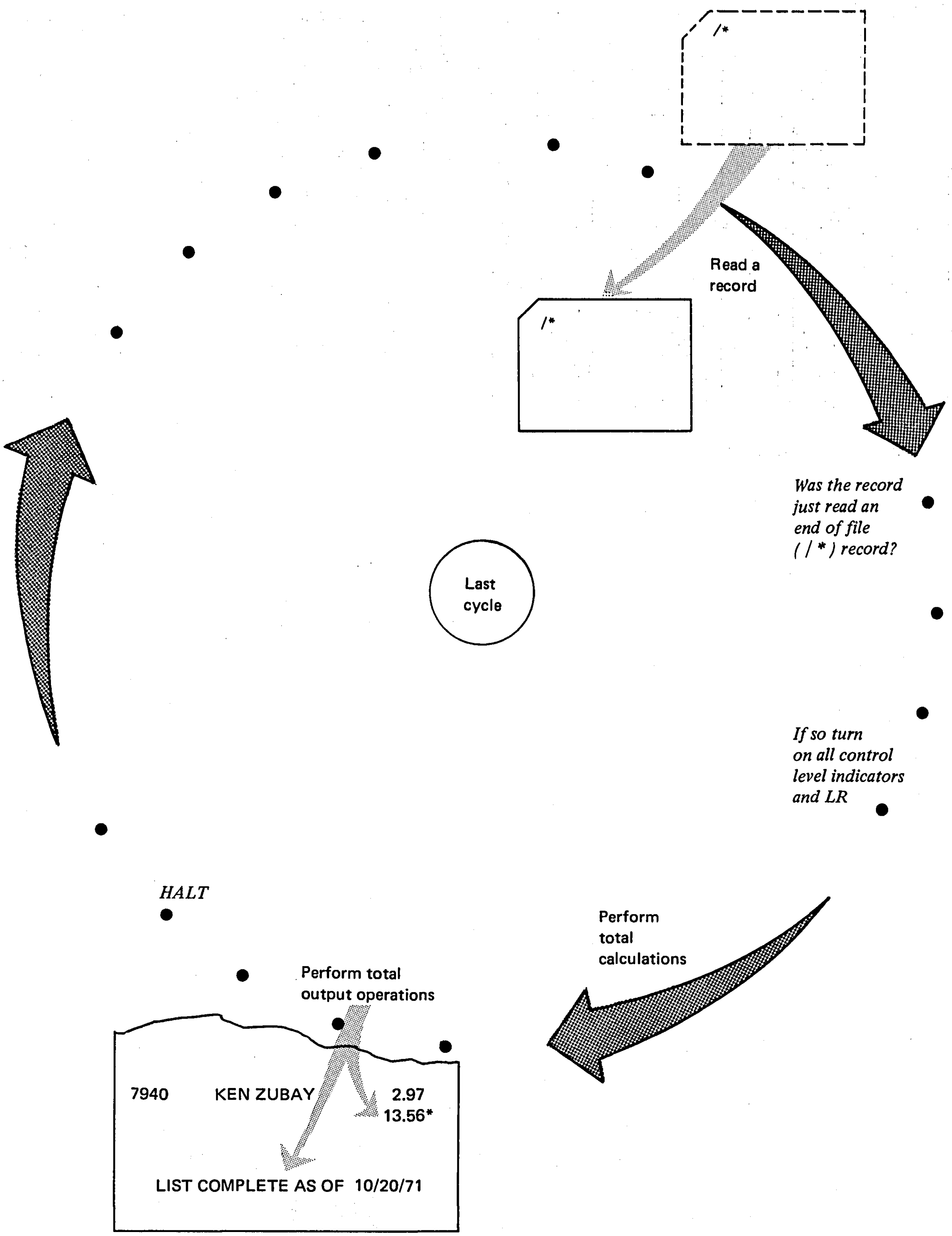
Programmer | Date

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O								
- -	-								
07	O								
08	O							51	'*'
09	O		T	3	LR				
10	O							53	'LIST COMPLETE AS OF'
11	O					UDATE	Y	62	
12	O								

Figure 1-12. Total Lines Conditioned by LR


/*
/*
Read a record
Was the record just read an end of file (/ *) record?
Last cycle
If so turn on all control level indicators and LR
Perform total calculations
Perform total output operations
HALT

```
7940      KEN ZUBAY          2.97
                            13.56*

     LIST COMPLETE AS OF  10/20/71
```

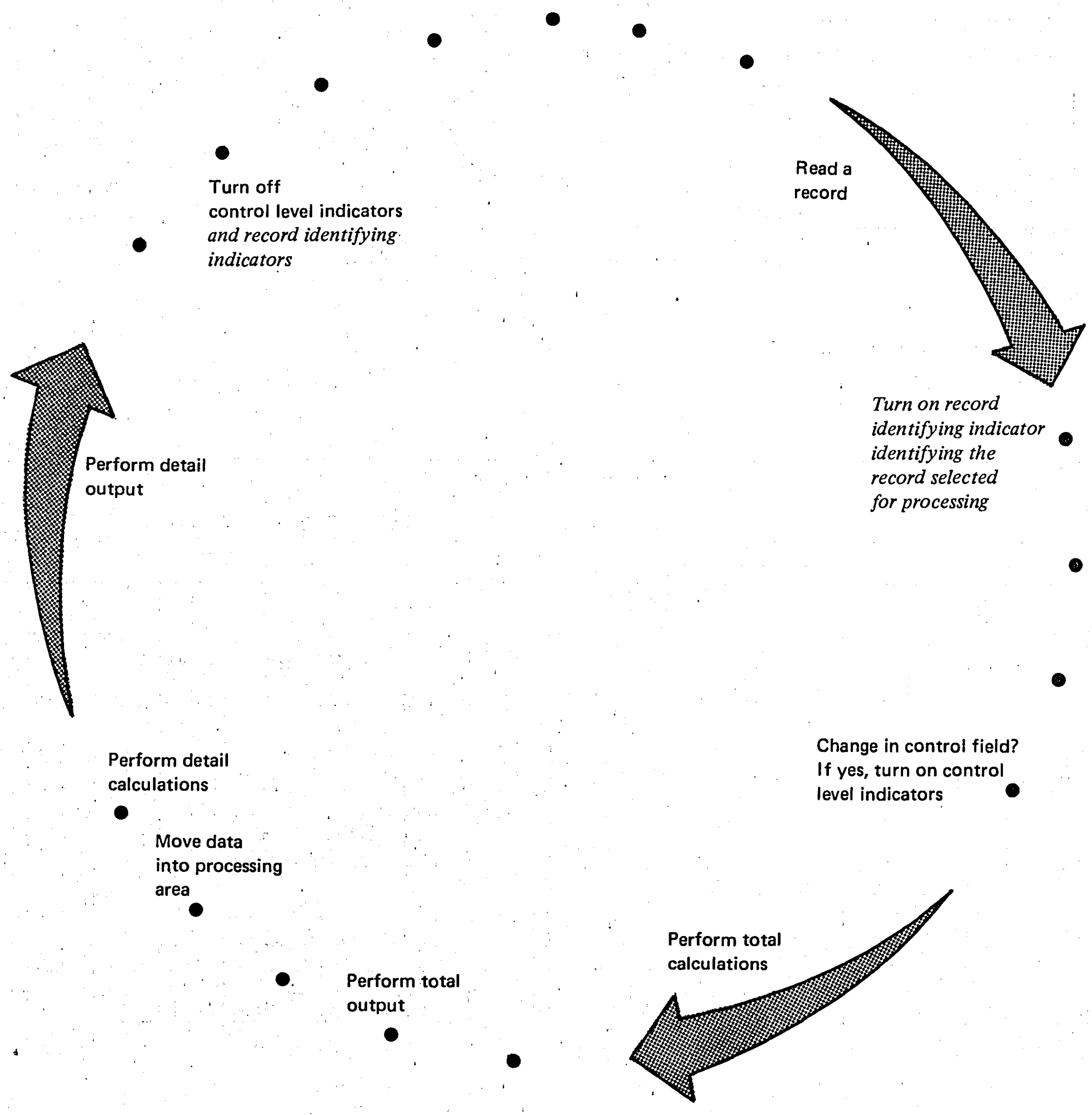
Figure 1-13. Program Cycle Illustrating the LR Indicator

Record Identifying Indicators (01-99)

You assign a record identifying indicator to each type of record in the input file. If certain operations are to be performed for one record type only, you may condition those operations by the appropriate record identifying indicator. By this method you can tell the RPG II program what operations to perform when it processes a specific record type.

After the program has selected the next record to process, it turns on the record identifying indicator which you assigned to that record type. This indicator is turned off only at the end of each program cycle; thus it is on during both detail and total operations. Detail and total operations conditioned by the record identifying indicator, currently on, will then be performed.

Figure 1-14 shows specific steps in the RPG II logic related to record identifying indicators.



Read a record
Turn on record identifying indicator identifying the record selected for processing
Change in control field? If yes, turn on control level indicators
Perform total calculations
Perform total output
Move data into processing area
Perform detail calculations
Perform detail output
Turn off control level indicators *and record identifying indicators*

Figure 1-14. Logic for Record Identifying Indicators

Consider the use of record identifying indicators in a billing job. A monthly file is kept which contains records of purchases and payments made by each customer. In addition, the file contains a balance forward record for each customer. Figure 1-15 shows the three input record types used and output records required.

The three record types are defined on the input sheet. Each type is given a different record identifying indicator. The record identifying indicators are then used to indicate which operations are to be performed for each record type. Figure 1-16 shows the input, calculation, and output-format specifications for the job. Use these specifications to help you follow, step by step, the operations performed in the program cycles shown in Figure 1-17.

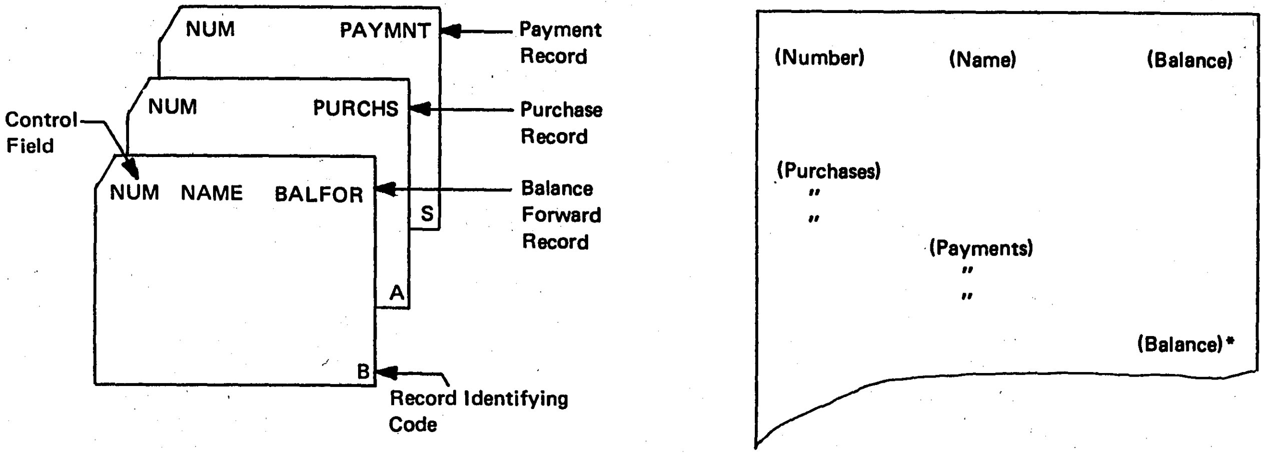
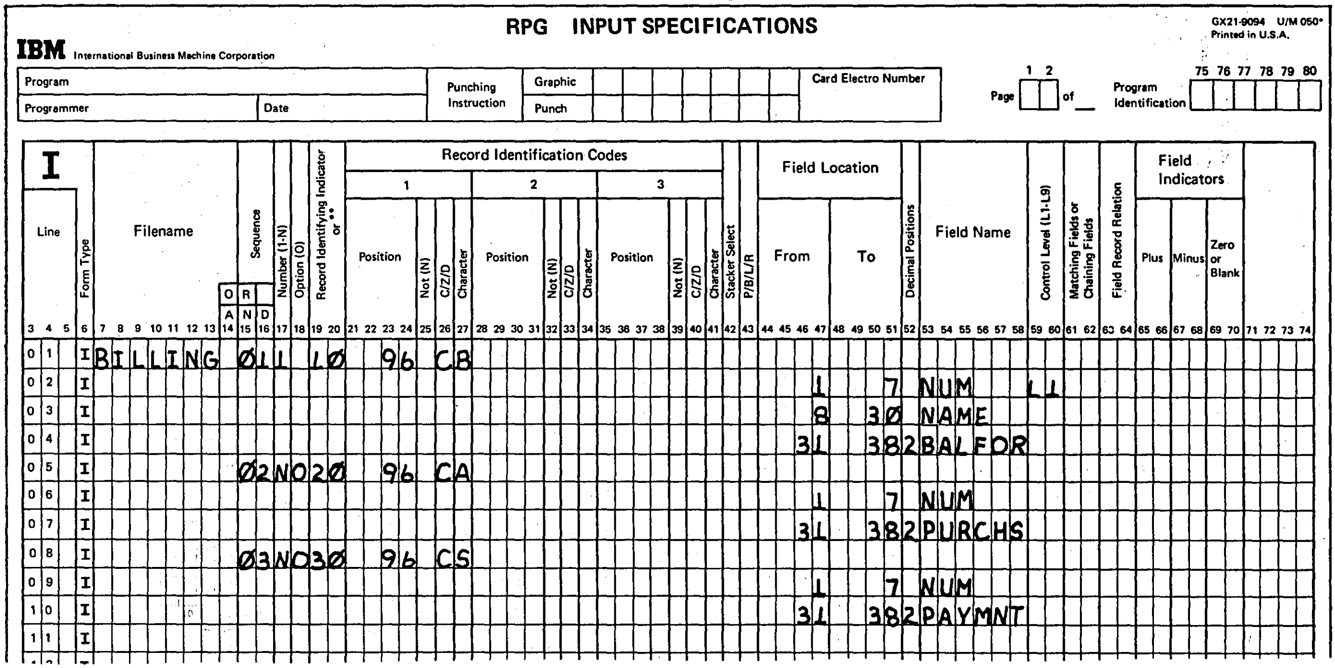


Figure 1-15. Input and Output for Billing Job



IBM International Business Machine Corporation

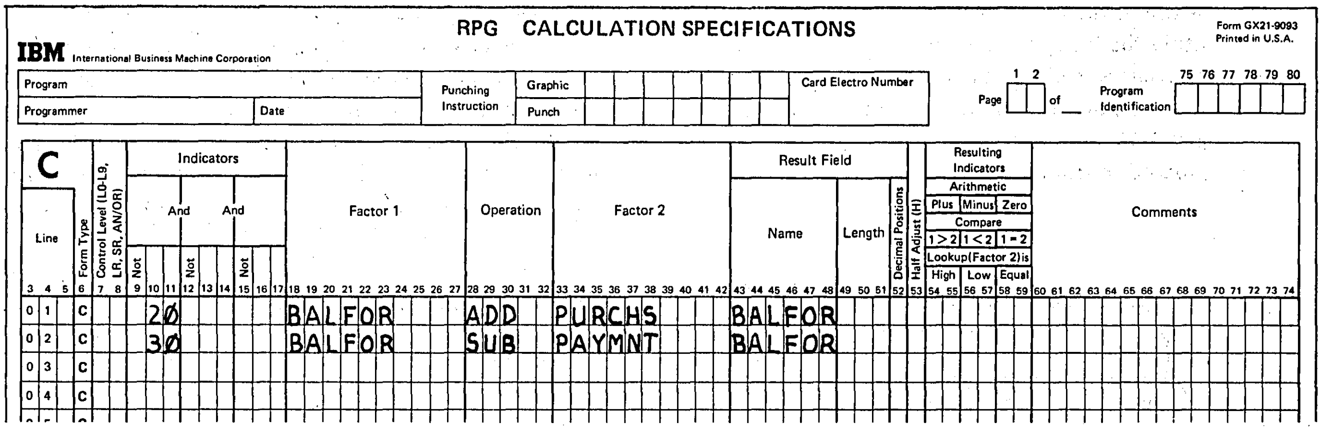
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 □□ of __ | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1 Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2 Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3 Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P/B/L/R (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	(71-74)
01	I	BILLING	Ø1	1		1Ø	96		C	B																					
02	I																				1	7		NUM	L1						
03	I																				8	3Ø		NAME							
04	I																				31	38	2	BALFOR							
05	I		Ø2	N	O	2Ø	96		C	A																					
06	I																				1	7		NUM							
07	I																				31	38	2	PURCHS							
08	I		Ø3	N	O	3Ø	96		C	S																					
09	I																				1	7		NUM							
10	I																				31	38	2	PAYMNT							
11	I																														

Figure 1-16 (Part 1 of 2). Billing Job Specifications Using Record Identifying Indicators



IBM International Business Machine Corporation

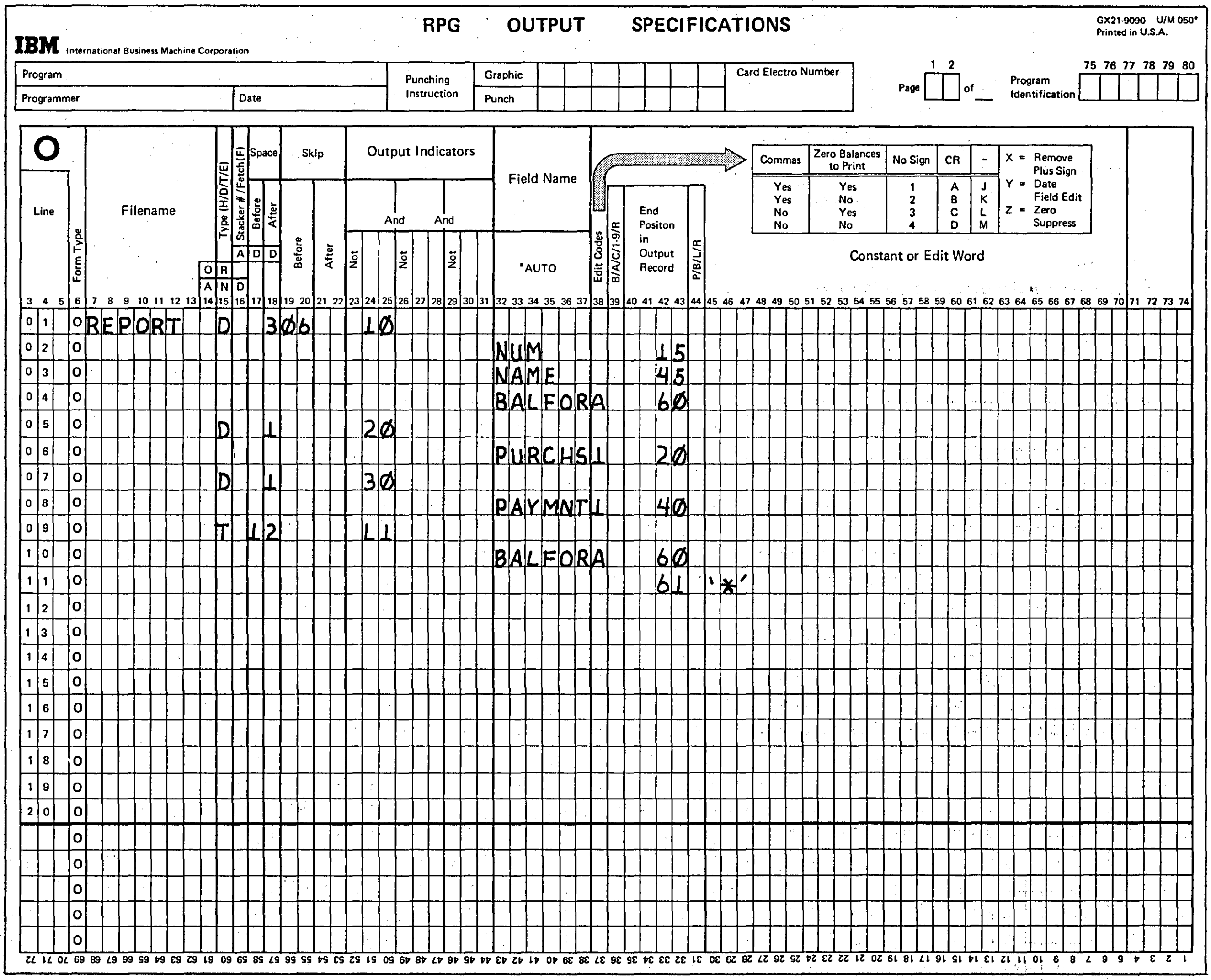
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

C — Line | Form Type | Control Level (L0-L9, LR, SR, AN/OR) | Indicators (Not, And, Not, And, Not) | Factor 1 | Operation | Factor 2 | Result Field (Name, Length) | Decimal Positions | Half Adjust (H) | Resulting Indicators (Arithmetic: Plus, Minus, Zero; Compare: 1>2, 1<2, 1=2; Lookup(Factor 2)is: High, Low, Equal) | Comments

```
  01 C   2Ø      BALFOR    ADD  PURCHS    BALFOR
  02 C   3Ø      BALFOR    SUB  PAYMNT    BALFOR
  03 C
  04 C
```



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

O — Line | Form Type | Filename | Type (H/D/T/E) | Stacker #/Fetch(F) | Space (Before, After) | Skip (Before, After) | Output Indicators (Not, And, Not, And, Not) | Field Name (*AUTO) | Edit Codes | B/A/C/1-9/R | End Positon in Output Record | P/B/L/R | Constant or Edit Word

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

```
  01 OREPORT  D  3Ø6   1Ø
  02 O                         NUM       15
  03 O                         NAME      45
  04 O                         BALFORA   6Ø
  05 O        D  1     2Ø
  06 O                         PURCHS1   2Ø
  07 O        D  1     3Ø
  08 O                         PAYMNT1   4Ø
  09 O        T 12     L1
  10 O                         BALFORA   6Ø
  11 O                                   61 '*'
  12 O
  13 O
  14 O
  15 O
  16 O
  17 O
  18 O
  19 O
  20 O
     O
     O
     O
     O
     O
```

Figure 1-16 (Part 2 of 2). Billing Job Specifications Using Record Identifying Indicators

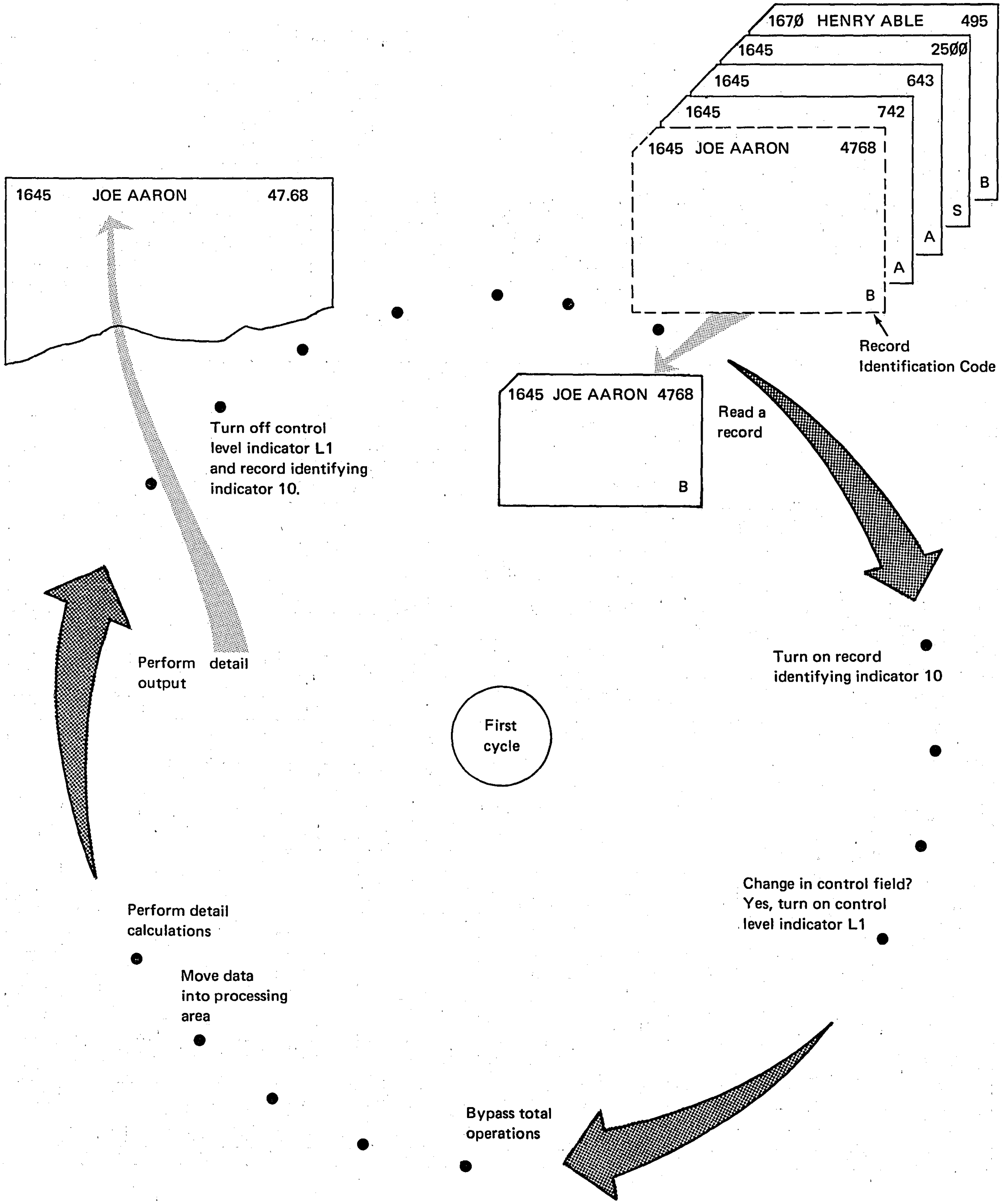

1670 HENRY ABLE 495
1645 2500
1645 643
1645 742
1645 JOE AARON 4768
B
S
A
A
B
Record Identification Code
1645 JOE AARON 4768
B
Read a record
1645 JOE AARON 47.68
Turn off control level indicator L1 and record identifying indicator 10.
Turn on record identifying indicator 10
First cycle
Perform detail output
Change in control field? Yes, turn on control level indicator L1
Perform detail calculations
Move data into processing area
Bypass total operations

Figure 1-17 (Part 1 of 3). Program Cycle Illustrating Use of Record Identifying Indicators

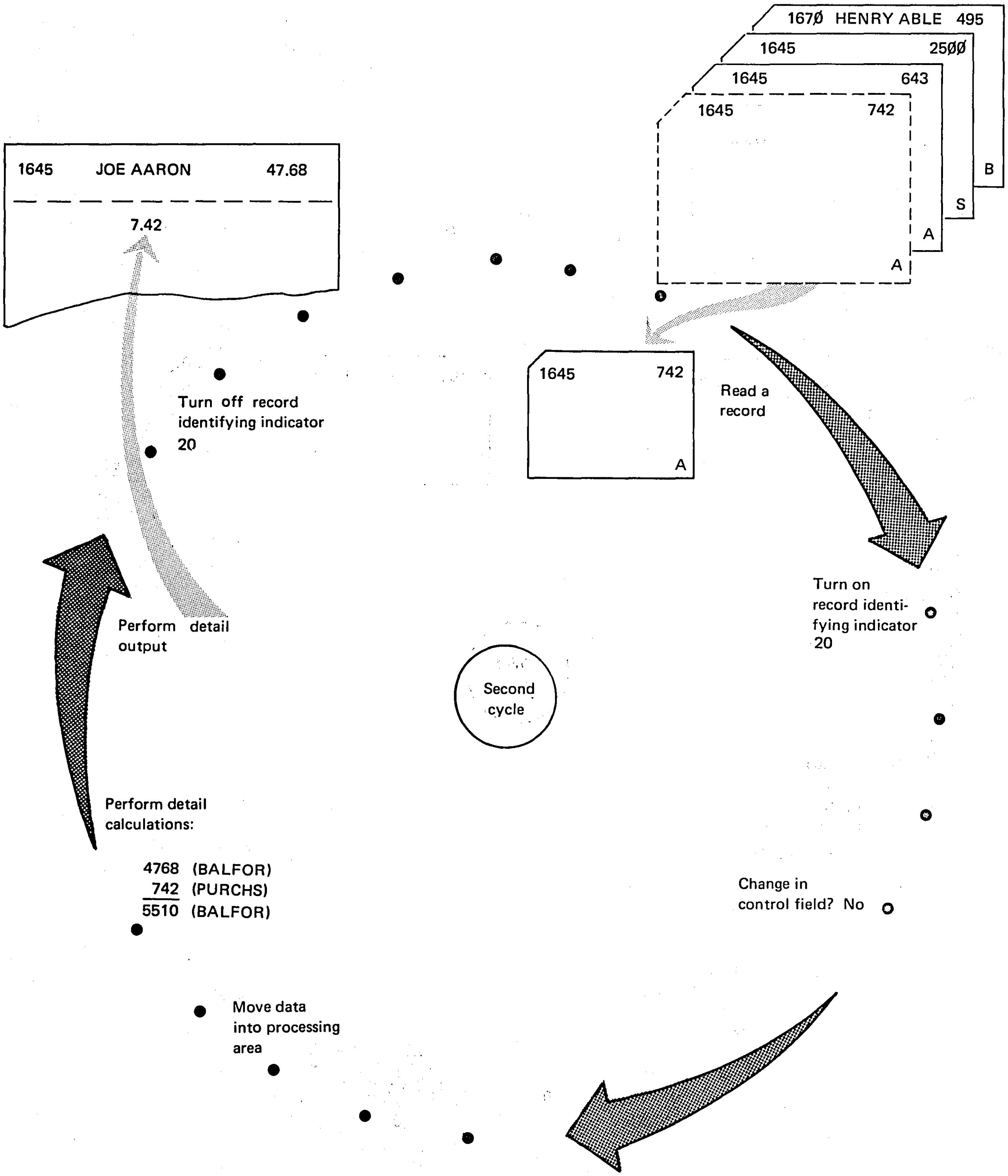


Figure 1-17 (Part 2 of 3). Program Cycle Illustrating Use of Record Identifying Indicators

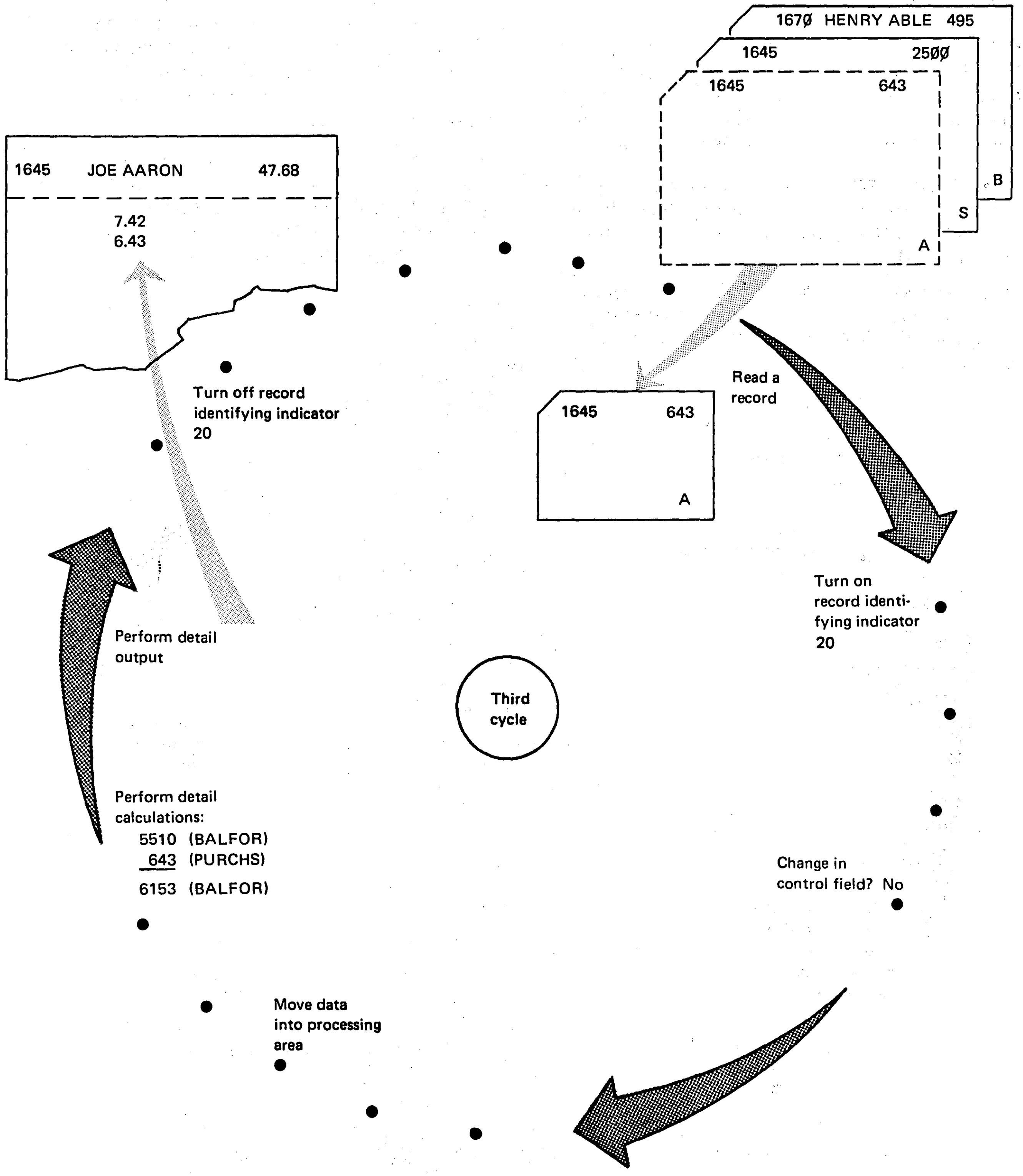

1670 HENRY ABLE 495
1645 2500
1645 643
B
S
A
1645 JOE AARON 47.68
7.42
6.43
Read a record
1645 643
A
Turn off record identifying indicator 20
Turn on record identifying indicator 20
Perform detail output
Third cycle
Perform detail calculations:
5510 (BALFOR)
643 (PURCHS)
6153 (BALFOR)
Change in control field? No
Move data into processing area

Figure 1-17 (Part 3 of 3). Program Cycle Illustrating Use of Record Identifying Indicators

Field Indicators (01-99)

Field indicators are used to test a field on an input record for a plus, minus, zero or blank value. Any operations that are to be performed only when a numeric field is plus, minus, or zero or when an alphameric field is blank may be conditioned by the appropriate field indicator.

Note: A numeric field that is all blanks will turn on an indicator specified for all zeros. However, if an alphameric field is all zeros, the field will not turn on an indicator specified for all blanks.

Field indicators are turned on or off after data from the record to be processed has been moved into the processing area. Figure 1-18 shows the RPG II logic related to field indicators.

For each program cycle, field indicators are set to reflect the result of the test on a field. If the condition tested for is satisfied, they are turned on; if the condition is not satisfied, they are turned off. A field indicator that is set as the result of a test retains its setting until another test is made using the same indicator.

When the indicator is on, any detail and total operations conditioned by the field indicator may be performed before testing of a field again resets the indicator. Remember that at total time, however, the field indicator will have the setting established in the *previous* cycle.

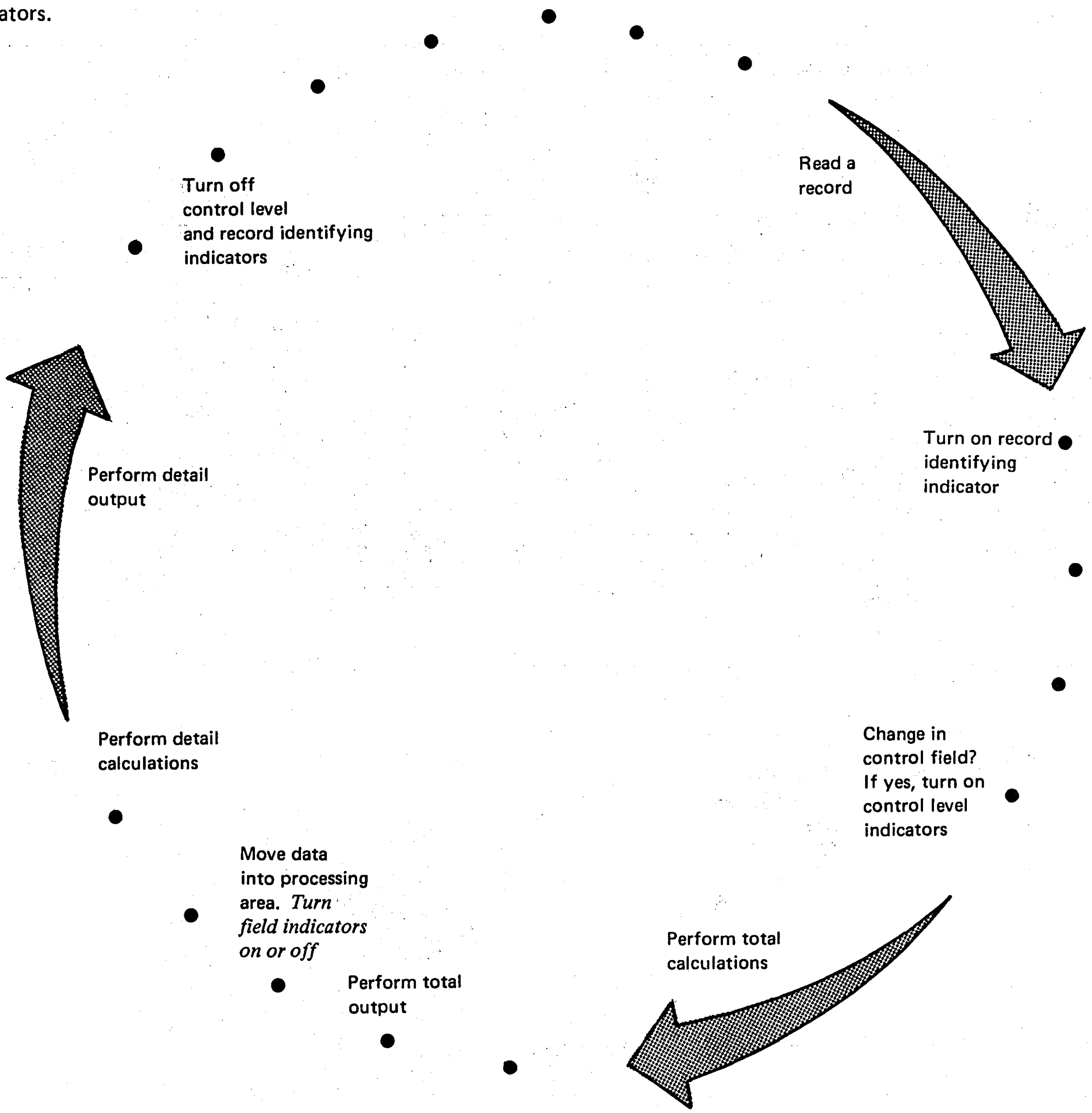

Read a record
Turn on record identifying indicator
Change in control field? If yes, turn on control level indicators
Perform total calculations
Perform total output
Move data into processing area. *Turn field indicators on or off*
Perform detail calculations
Perform detail output
Turn off control level and record identifying indicators

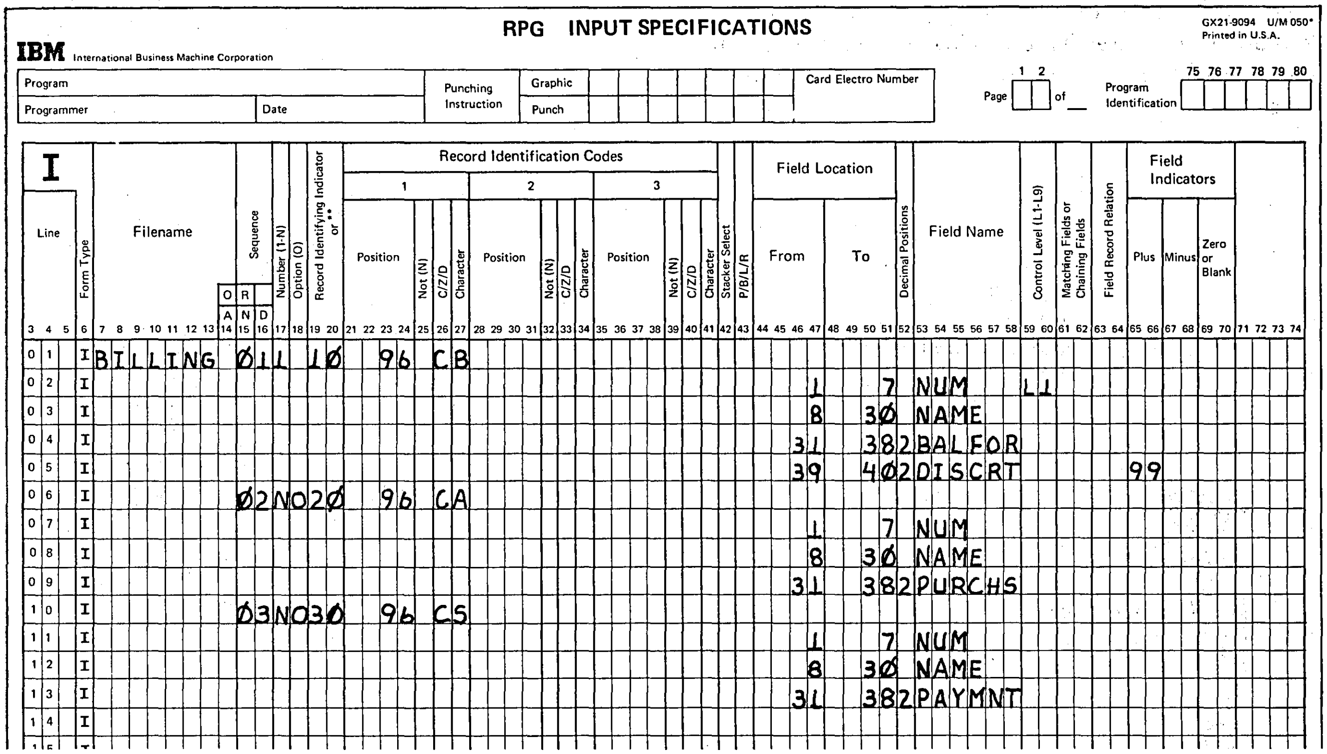
Figure 1-18. Logic for Field Indicators

Consider the use of field indicators in the billing job previously described. The same record types are used as in the previous job. The only difference is the additional field, discount (DISCRT), in columns 39-40 on the balance forward record.

All employees receive a discount on everything they buy. The rate of discount they receive is recorded in the DISCRT field. All accounts other than employee accounts have a zero in the discount field since they receive no discount.

Each time a record with a B in position 96 is read, DISCRT must be tested. Only when it contains a positive value should discount be calculated. Figure 1-19 shows the input, calculation, and output-format specifications for this job.

Use these specifications to help you follow, step by step, the logic for each program cycle shown in Figure 1-20.



IBM International Business Machine Corporation

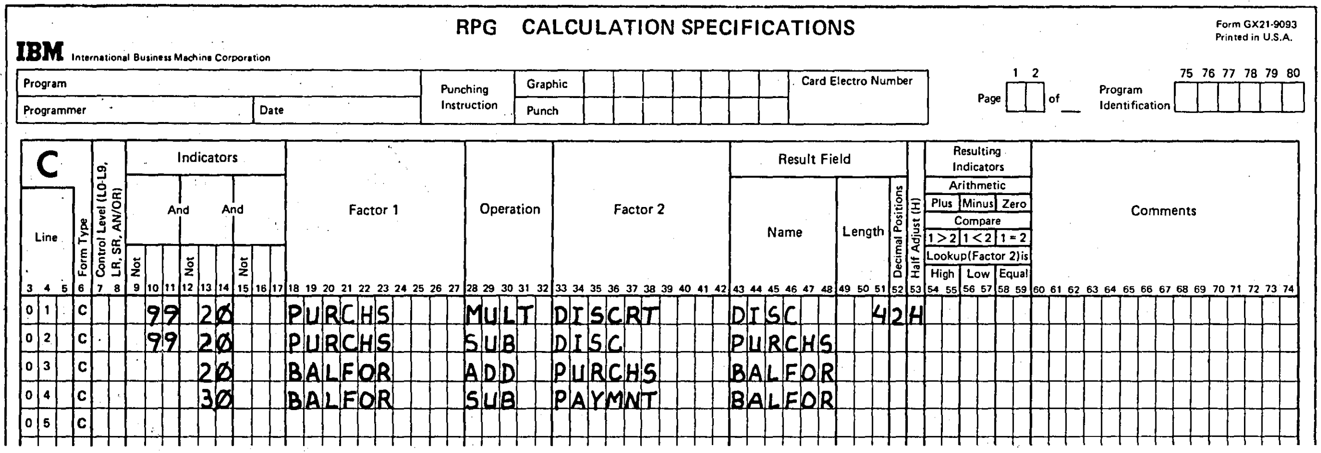
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page □□ of __
Program Identification □□□□□□

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank
01	I	BILLING	Ø1	1		1Ø	96		C	B										
02	I										1	7		NUM	L1					
03	I										8	3Ø		NAME						
04	I										31	38	2	BALFOR						
05	I										39	4Ø	2	DISCRT				99		
06	I		Ø2	N	O	2Ø	96		C	A										
07	I										1	7		NUM						
08	I										8	3Ø		NAME						
09	I										31	38	2	PURCHS						
10	I		Ø3	N	O	3Ø	96		C	S										
11	I										1	7		NUM						
12	I										8	3Ø		NAME						
13	I										31	38	2	PAYMNT						
14	I																			

Figure 1-19 (Part 1 of 2). Billing Job Specifications Using Field Indicators



IBM International Business Machine Corporation

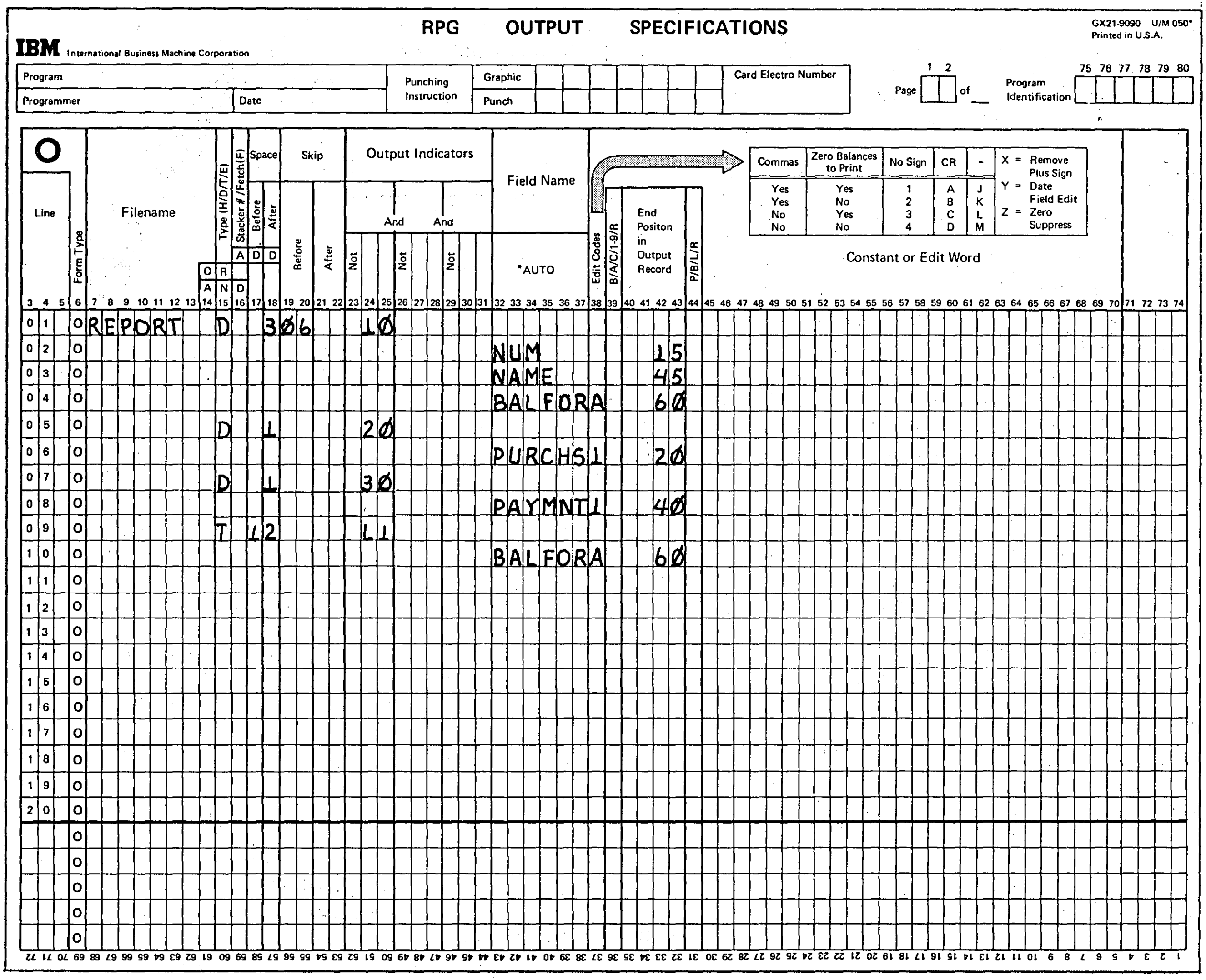
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Programmer | Date

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not (9)	Indicators (10-11)	And: Not (12)	And (13-14)	And: Not (15)	And (16-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators Arithmetic Plus / Compare 1>2 / Lookup(Factor 2)is High (54-55)	Minus / 1<2 / Low (56-57)	Zero / 1=2 / Equal (58-59)	Comments (60-74)
01	C			99		2Ø			PURCHS	MULT	DISCRT	DISC	4	2	H				
02	C			99		2Ø			PURCHS	SUB	DISC	PURCHS							
03	C					2Ø			BALFOR	ADD	PURCHS	BALFOR							
04	C					3Ø			BALFOR	SUB	PAYMNT	BALFOR							
05	C																		



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Programmer | Date

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-13)	Type (H/D/T/E) (15)	Stacker #/Fetch(F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators Not (23)	(24-25)	And Not (26)	(27-28)	And Not (29)	(30-31)	Field Name / *AUTO (32-37)	Edit Codes (38)	B/A/C/1-9/R (39)	End Positon in Output Record (40-43)	P/B/L/R (44)	Constant or Edit Word (45-70)
01	O	REPORT	D			3	Ø6			1Ø										
02	O														NUM			15		
03	O														NAME			45		
04	O														BALFOR	A		6Ø		
05	O		D			1				2Ø										
06	O														PURCHS	1		2Ø		
07	O		D			1				3Ø										
08	O														PAYMNT	1		4Ø		
09	O		T		1	2				L1										
10	O														BALFOR	A		6Ø		
11	O																			
12	O																			
13	O																			
14	O																			
15	O																			
16	O																			
17	O																			
18	O																			
19	O																			
20	O																			
	O																			
	O																			
	O																			
	O																			
	O																			

Figure 1-19 (Part 2 of 2). Billing Job Specifications Using Field Indicators

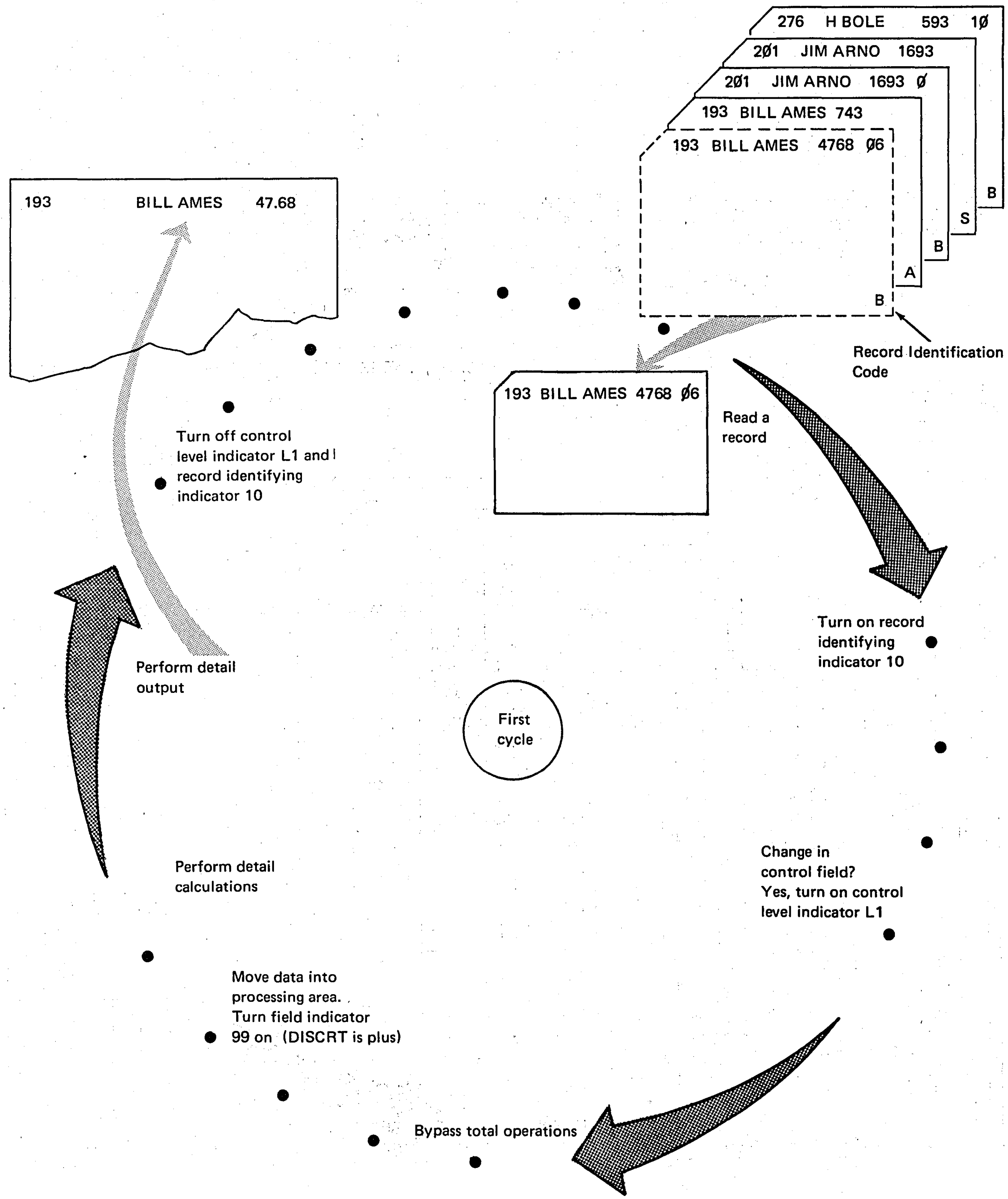

276 H BOLE 593 1Ø
2Ø1 JIM ARNO 1693
2Ø1 JIM ARNO 1693 Ø
193 BILL AMES 743
193 BILL AMES 4768 Ø6
B
S
B
A
B
Record Identification Code
193 BILL AMES 47.68
193 BILL AMES 4768 Ø6
Read a record
Turn on record identifying indicator 10
Change in control field? Yes, turn on control level indicator L1
Bypass total operations
Move data into processing area. Turn field indicator 99 on (DISCRT is plus)
Perform detail calculations
Perform detail output
Turn off control level indicator L1 and record identifying indicator 10
First cycle

Figure 1-20 (Part 1 of 3). Program Cycles Illustrating Use of Field Indicators

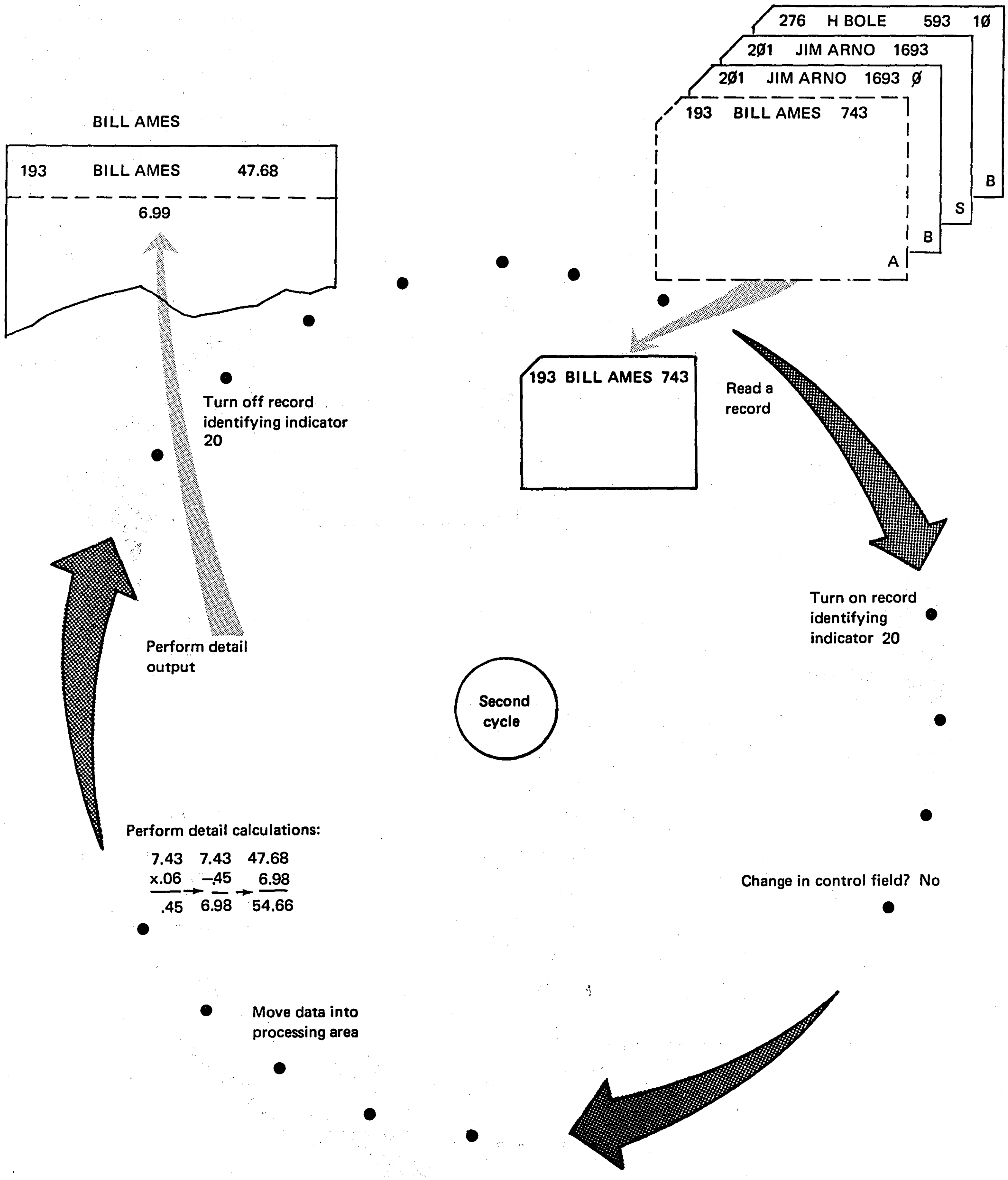


Figure 1-20 (Part 2 of 3). Program Cycles Illustrating Use of Field Indicators

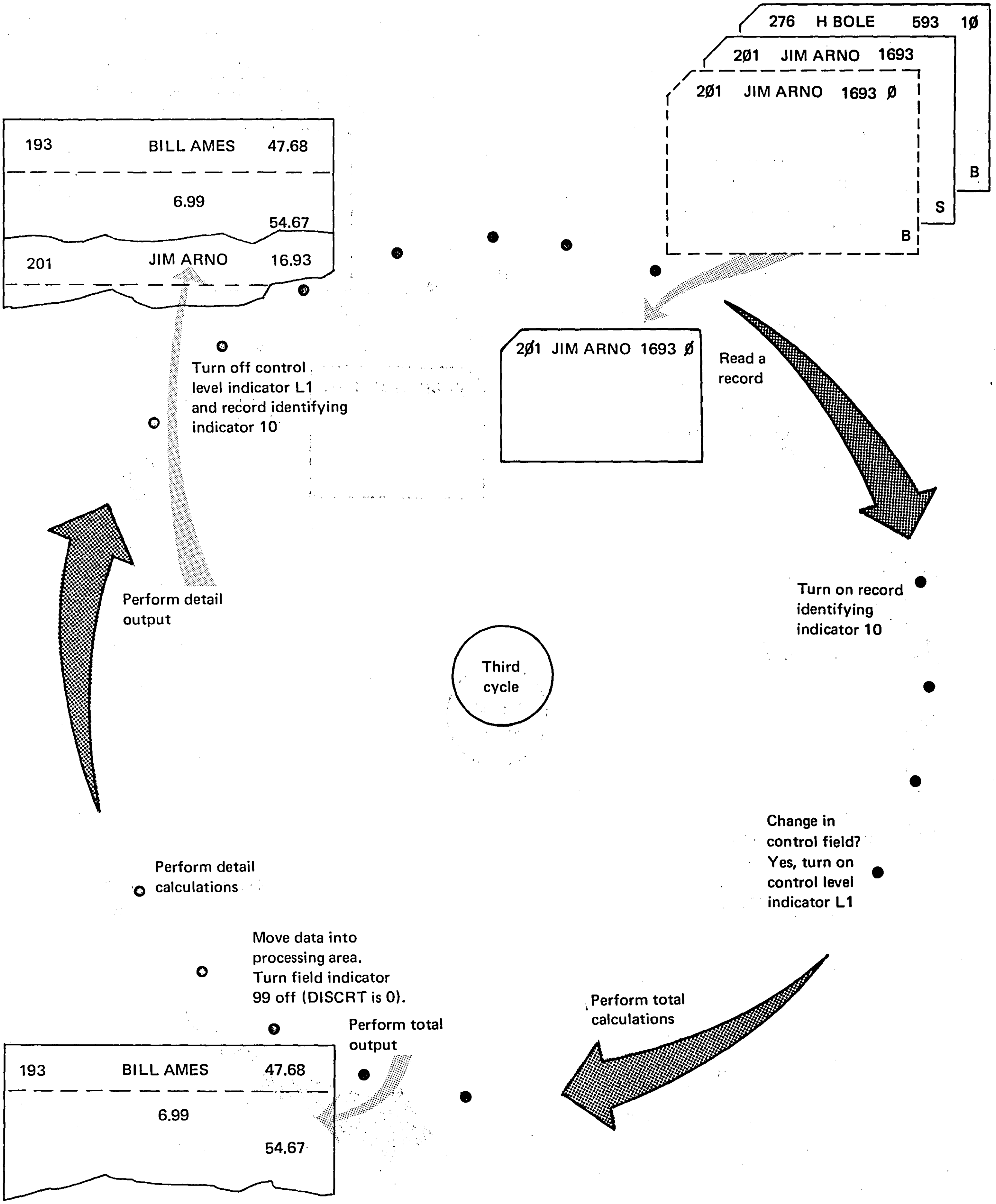

276 H BOLE 593 1Ø
2Ø1 JIM ARNO 1693
2Ø1 JIM ARNO 1693 Ø
B
S
B
193 BILL AMES 47.68
6.99
54.67
201 JIM ARNO 16.93
2Ø1 JIM ARNO 1693 Ø
Read a record
Turn off control level indicator L1 and record identifying indicator 10
Turn on record identifying indicator 10
Perform detail output
Third cycle
Change in control field? Yes, turn on control level indicator L1
Perform detail calculations
Move data into processing area. Turn field indicator 99 off (DISCRT is 0).
Perform total calculations
Perform total output
193 BILL AMES 47.68
6.99
54.67

Figure 1-20 (Part 3 of 3). Program Cycles Illustrating Use of Field Indicators

Resulting Indicators (01-99)

Resulting indicators are assigned to signal something about the result of a calculation operation. Any operation which is dependent upon the result of the calculation can then be conditioned by a resulting indicator.

Resulting indicators may be turned on or off at either detail or total calculation time. An indicator which is set as a result of the calculation operation retains this setting until the next time a calculation is done for which the same indicator is a resulting indicator and the condition is not satisfied. Figure 1-21 shows the RPG II logic related to resulting indicators.

A resulting indicator may change status in the same cycle. This happens when *one* indicator is assigned to signal the result of both a total and detail calculation. The total calculation could turn it off and the detail calculation could turn it on, or vice versa. The indicator will not, however, be reset to show that a field is blank or zero after being blanked out by the Blank After function (B in column 39 of the Output-Format sheet).

The use of resulting indicators is demonstrated by an inventory job which determines whether an item needs to be reordered. After inventory has been taken, the quantity on hand is recorded for each item. If the quantity on hand is 100 or less, reorder should be immediate. If the quantity

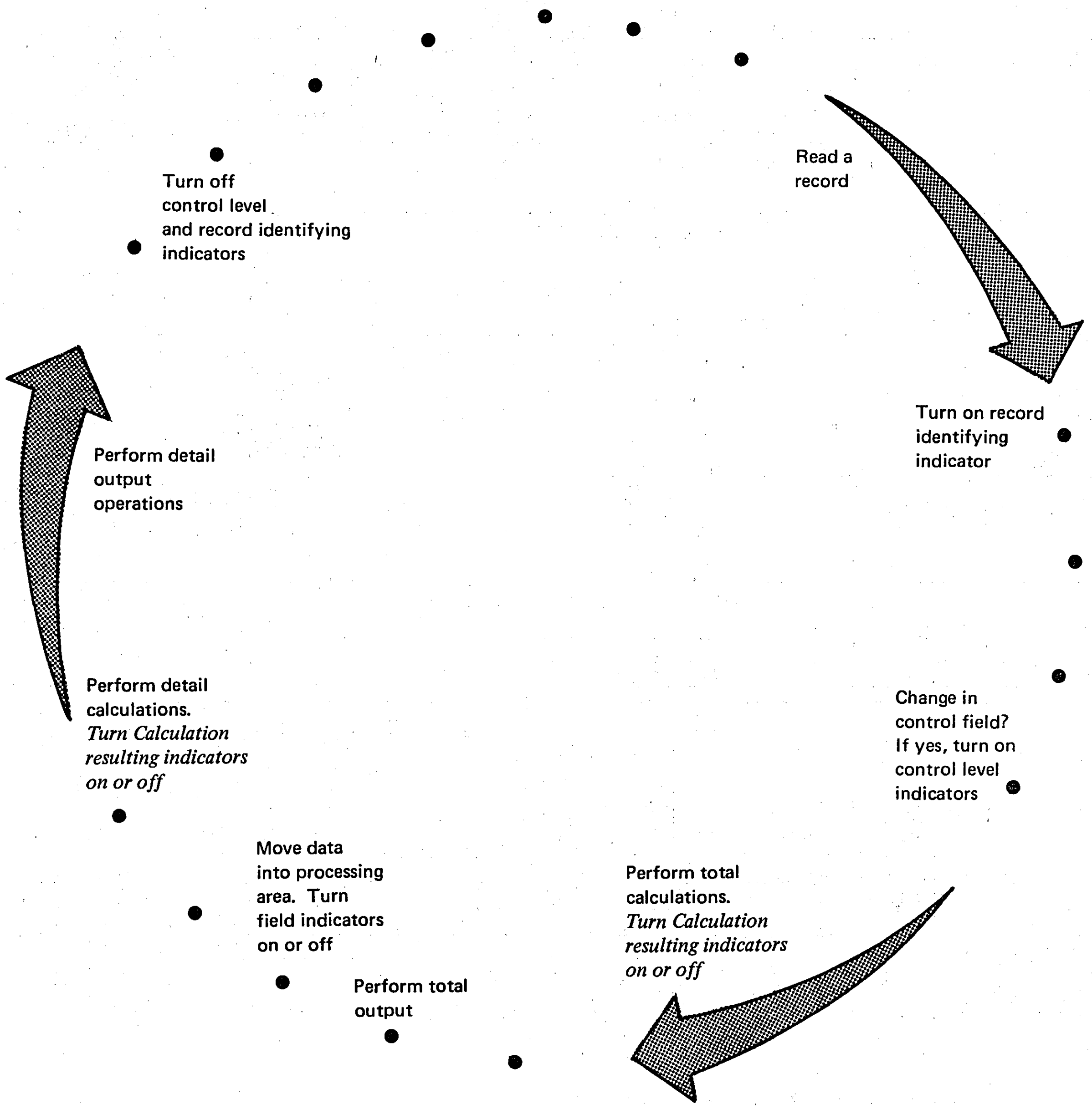
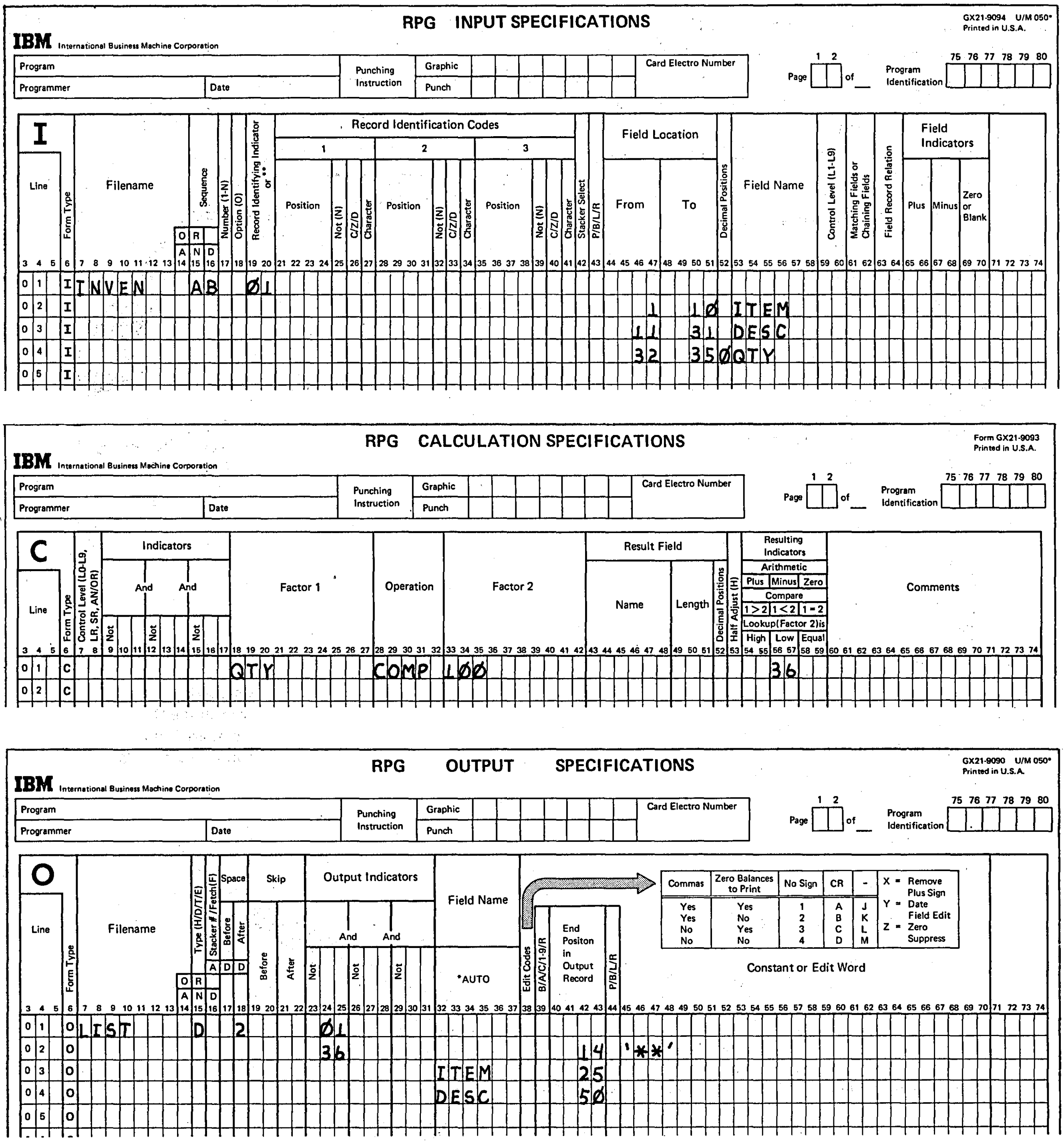

Read a record
Turn on record identifying indicator
Change in control field? If yes, turn on control level indicators
Perform total calculations. *Turn Calculation resulting indicators on or off*
Perform total output
Move data into processing area. Turn field indicators on or off
Perform detail calculations. *Turn Calculation resulting indicators on or off*
Perform detail output operations
Turn off control level and record identifying indicators

Figure 1-21. Logic for Resulting Indicators

is over 100, the item need not be reordered at this time. A list of all items is printed. All items to be reordered are indicated with a double asterisk. Figure 1-22 shows the specifications for the job. Use these specifications to help you follow the program cycles shown in Figure 1-23.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank
0 1	I	INVEN	AB			Ø1													
0 2	I									1	1Ø		ITEM						
0 3	I									11	31		DESC						
0 4	I									32	35	Ø	QTY						
0 5	I																		

IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High (Plus, 1>2)	Low (Minus, 1<2)	Equal (Zero, 1=2)	Comments
0 1	C			QTY	COMP	1ØØ						36		
0 2	C													

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Stacker #/Fetch(F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name *AUTO	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
0 1	O	LIST	D			2			Ø1						
0 2	O								36				14		'**'
0 3	O									ITEM			25		
0 4	O									DESC			5Ø		
0 5	O														

Figure 1-22. Inventory Job Specifications Using Resulting Indicators

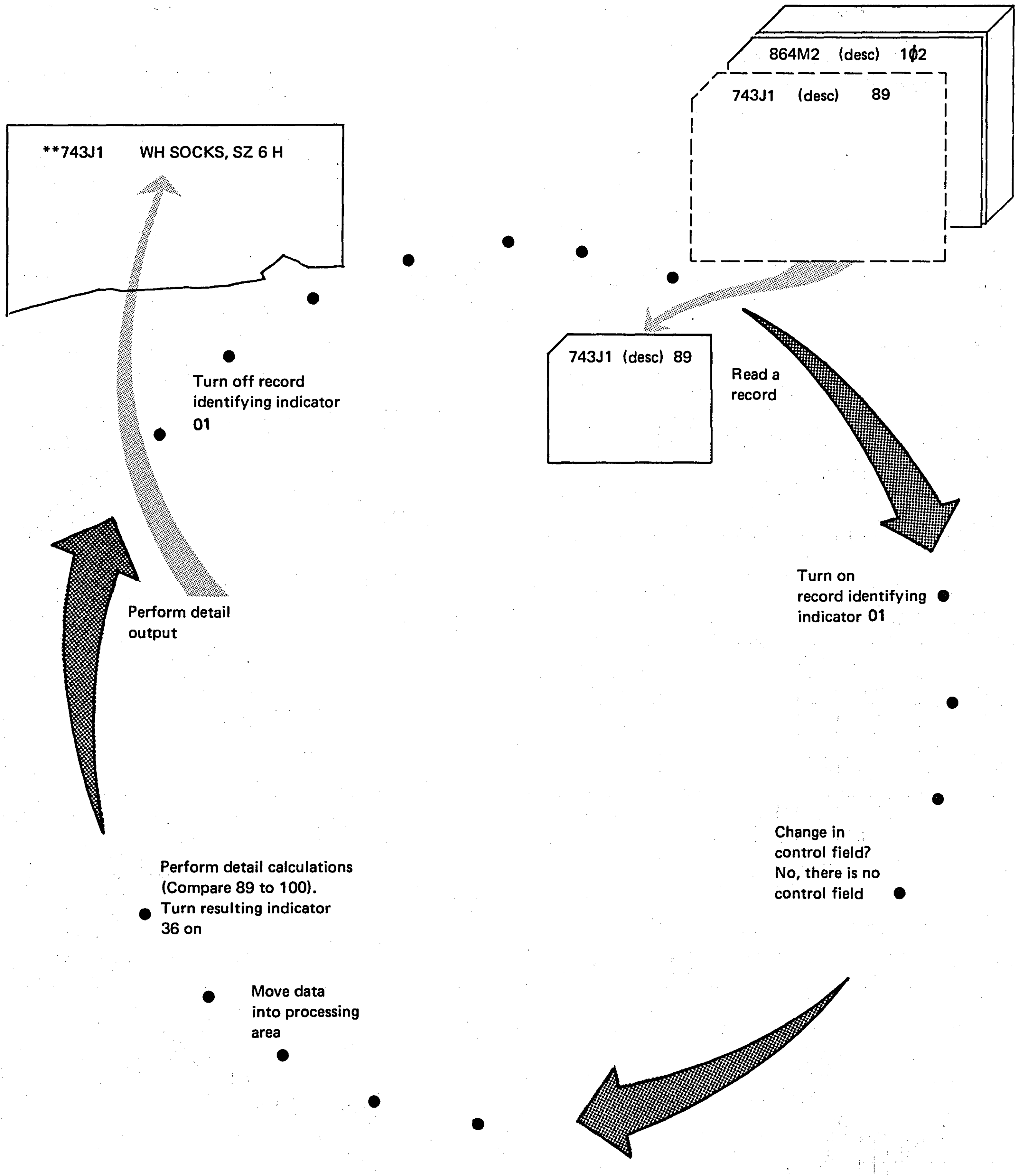

864M2 (desc) 1ϕ2
743J1 (desc) 89
743J1 (desc) 89
Read a record
Turn on record identifying indicator 01
Change in control field? No, there is no control field
Move data into processing area
Perform detail calculations (Compare 89 to 100). Turn resulting indicator 36 on
Perform detail output
Turn off record identifying indicator 01
**743J1 WH SOCKS, SZ 6 H

Figure 1-23 (Part 1 of 2). Program Cycles Illustrating Use of Resulting Indicators

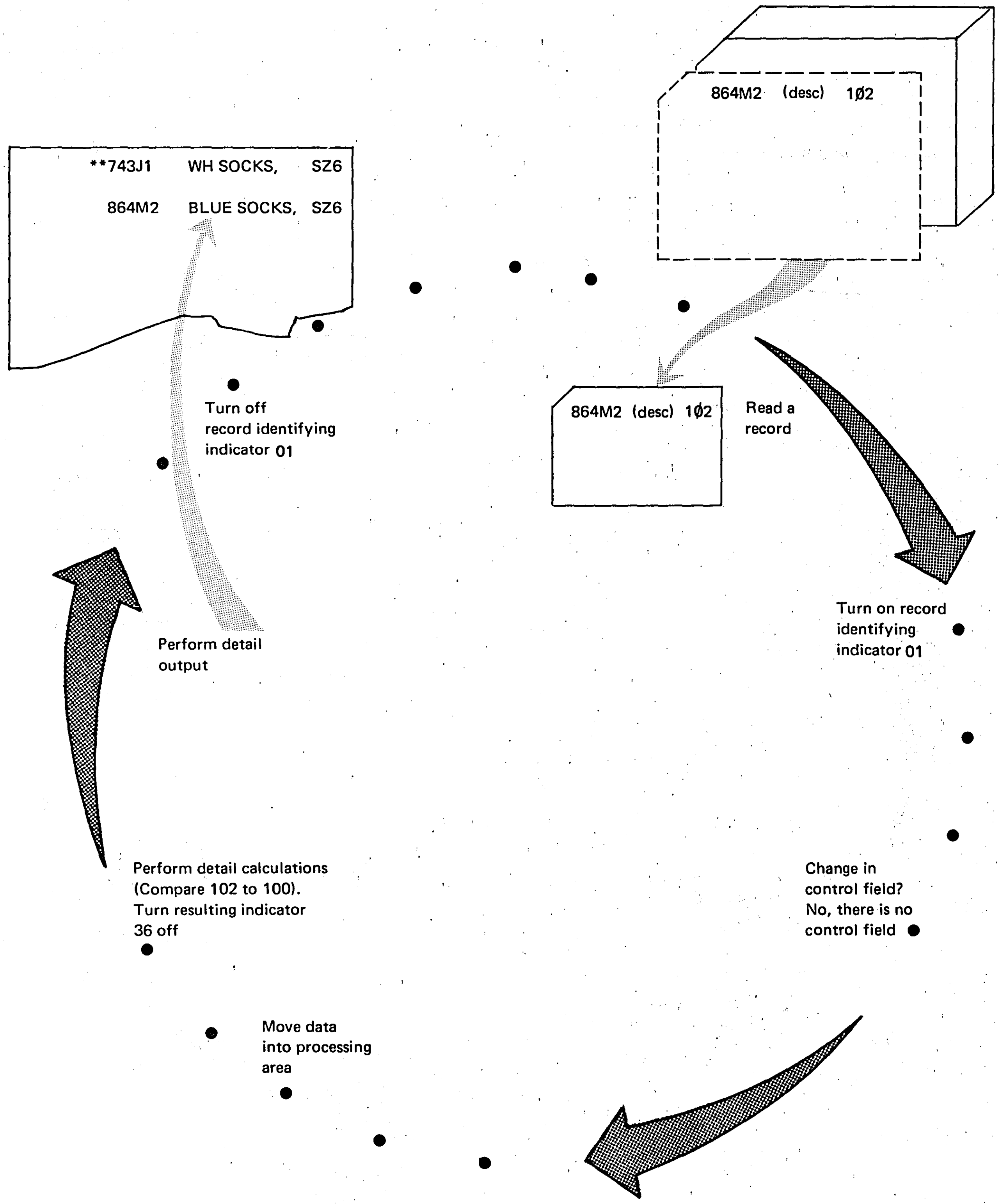

**743J1 WH SOCKS, SZ6
864M2 BLUE SOCKS, SZ6
864M2 (desc) 1Ø2
864M2 (desc) 1Ø2
Read a record
Turn on record identifying indicator 01
Change in control field? No, there is no control field
Move data into processing area
Perform detail calculations (Compare 102 to 100). Turn resulting indicator 36 off
Perform detail output
Turn off record identifying indicator 01

Figure 1-23 (Part 2 of 2). Program Cycles Illustrating Use of Resulting Indicators

Halt Indicators (H1-H9)

Halt indicators are used to stop the program when a specified condition is satisfied. Halt indicators may be used as record identifying, field, or resulting indicators. When halt indicators are used as record identifying indicators, a halt will be caused by a specific type of record; when used as field indicators, a halt will be caused by erroneous input data; when used as resulting indicators, a halt will be caused by erroneous results from calculations.

A halt indicator may be turned on at one of four different times (see Figure 1-24). Its use, of course, will determine when it is turned on. The program does not halt immediately when a halt indicator is turned on. All total and detail operations remaining in the cycle are performed first; then the program halts. This means that processing will still be completed on information from the record that caused the error condition.

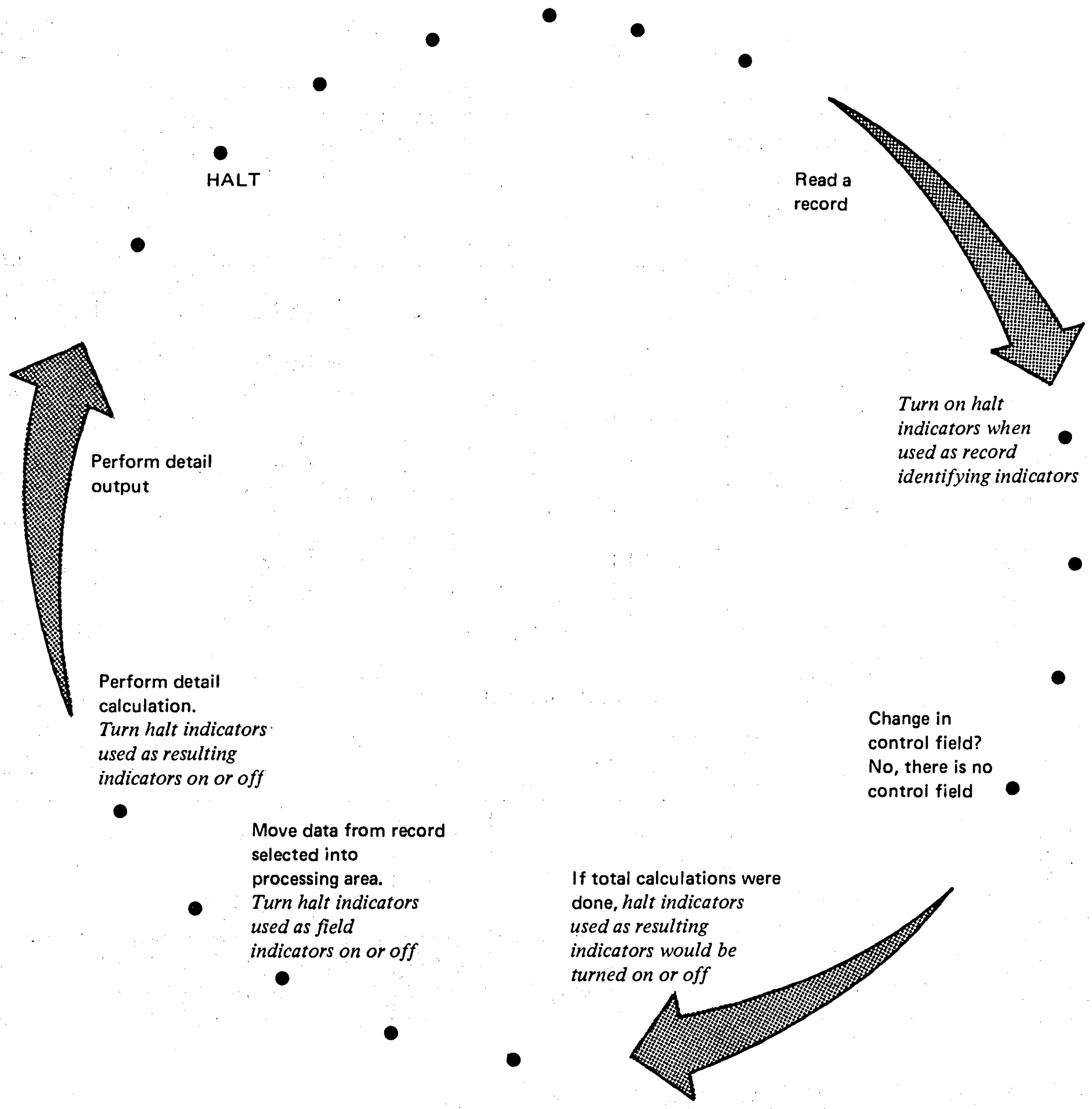
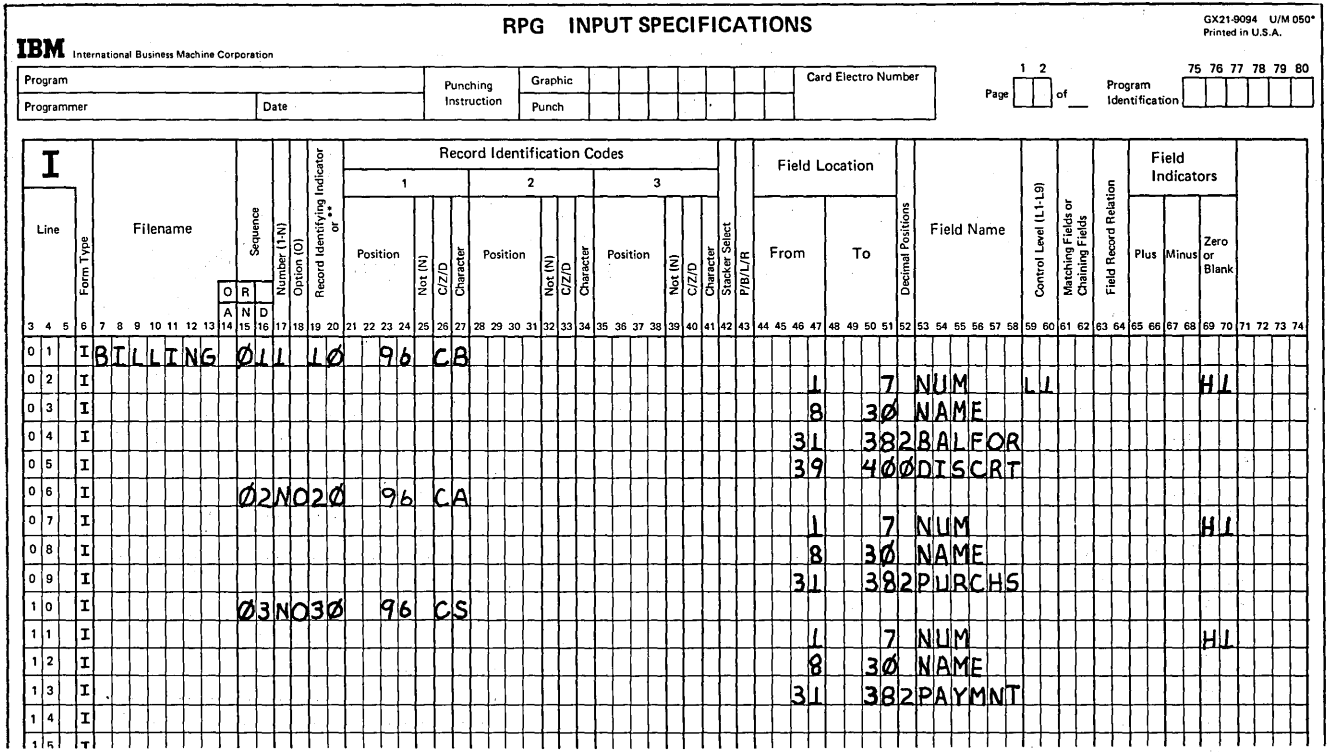

HALT
Read a record
Turn on halt indicators when used as record identifying indicators
Change in control field? No, there is no control field
If total calculations were done, *halt indicators used as resulting indicators would be turned on or off*
Move data from record selected into processing area. *Turn halt indicators used as field indicators on or off*
Perform detail calculation. *Turn halt indicators used as resulting indicators on or off*
Perform detail output

Figure 1-24. Logic for Halt Indicators

After a halt you may continue processing by pressing START on the processing unit. Halt indicators are always turned off before another program cycle begins.

Suppose a halt indicator were used in the billing job previously described in the discussion of field indicators. The halt indicator is used as a field indicator to check for an erro in the input record. When recording information for a customer who makes many purchases and payments, the NUM field is sometimes inadvertently omitted from the record. Any record with a blank NUM field should be corrected. Therefore, you must have some way of telling the computer to halt if the NUM field is blank. The indicator H1 in columns 69-70 (see Figure 1-25) will do this.



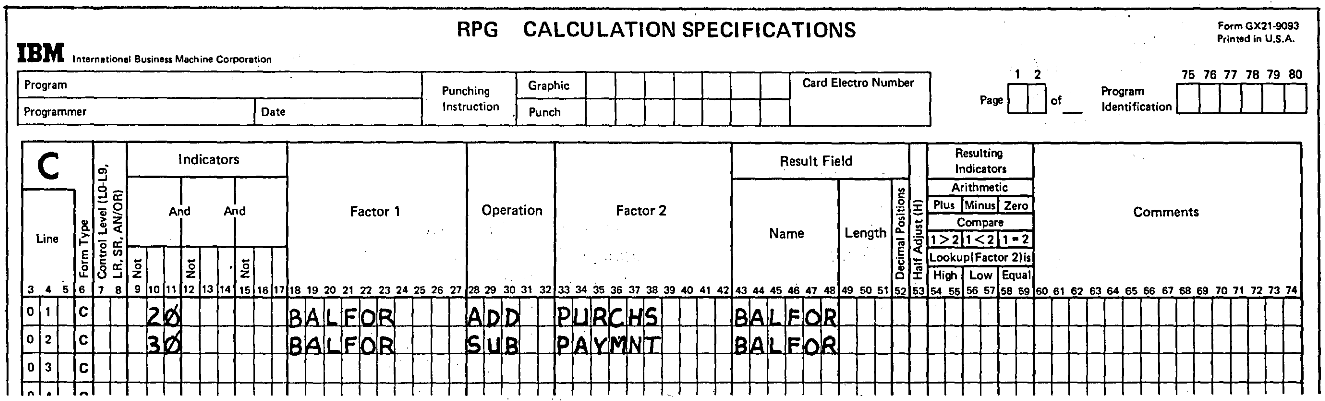
IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program — Programmer — Date — Punching Instruction: Graphic / Punch — Card Electro Number — Page of — Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Field Indicators Plus	Minus	Zero or Blank
01	I	BILLING	Ø1	1		1Ø	96		C	B								
02	I										1	7		NUM	L1			H1
03	I										8	3Ø		NAME				
04	I										31	38	2	BALFOR				
05	I										39	4Ø	Ø	DISCRT				
06	I		Ø2	N	O	2Ø	96		C	A								
07	I										1	7		NUM				H1
08	I										8	3Ø		NAME				
09	I										31	38	2	PURCHS				
10	I		Ø3	N	O	3Ø	96		C	S								
11	I										1	7		NUM				H1
12	I										8	3Ø		NAME				
13	I										31	38	2	PAYMNT				
14	I																	



IBM International Business Machine Corporation

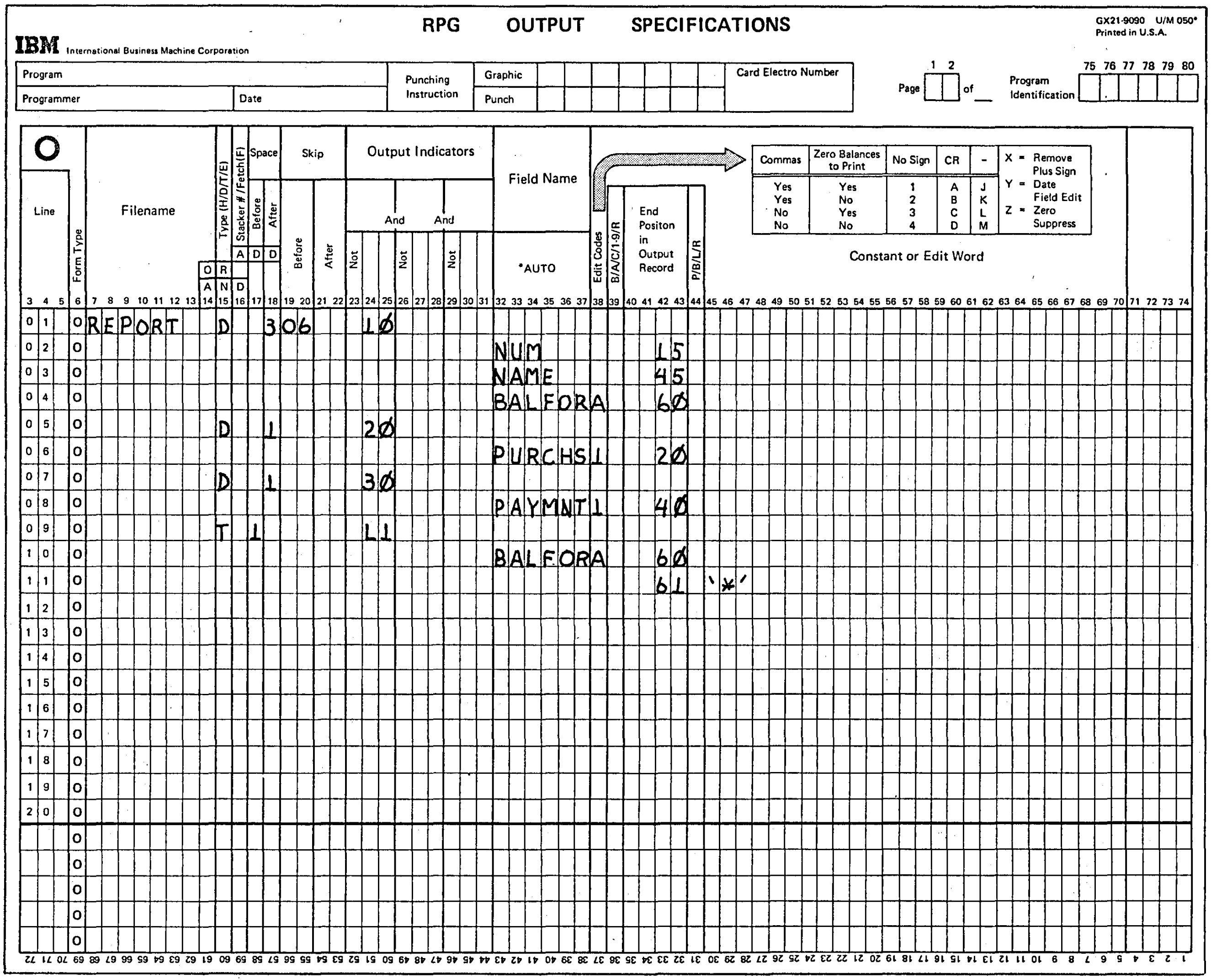
RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Program — Programmer — Date — Punching Instruction: Graphic / Punch — Card Electro Number — Page of — Program Identification

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	Resulting Indicators	Comments
01	C		2Ø	BALFOR	ADD	PURCHS	BALFOR					
02	C		3Ø	BALFOR	SUB	PAYMNT	BALFOR					
03	C											

Figure 1-25 (Part 1 of 2). Billing Job Specifications Using Halt Indicators



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program

Programmer Date

Punching Instruction | Graphic | Punch

Card Electro Number

Page of

Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker # / Fetch (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Positon in Output Record	P/B/L/R	Constant or Edit Word
01	O	REPORT	D			3	06		1Ø						
02	O									NUM			15		
03	O									NAME			45		
04	O									BALFORA			6Ø		
05	O		D			1			2Ø						
06	O									PURCHS1			2Ø		
07	O		D			1			3Ø						
08	O									PAYMNT1			4Ø		
09	O		T		1				L1						
10	O									BALFORA			6Ø		
11	O												61		'*'
12	O														
13	O														
14	O														
15	O														
16	O														
17	O														
18	O														
19	O														
20	O														
	O														
	O														
	O														
	O														
	O														

Figure 1-25 (Part 2 of 2). Billing Job Specifications Using Halt Indicators

Figure 1-26 shows the three program cycles. In the first cycle there is no error. In the second, a halt occurs because of a blank number field. The third begins with another record being read.

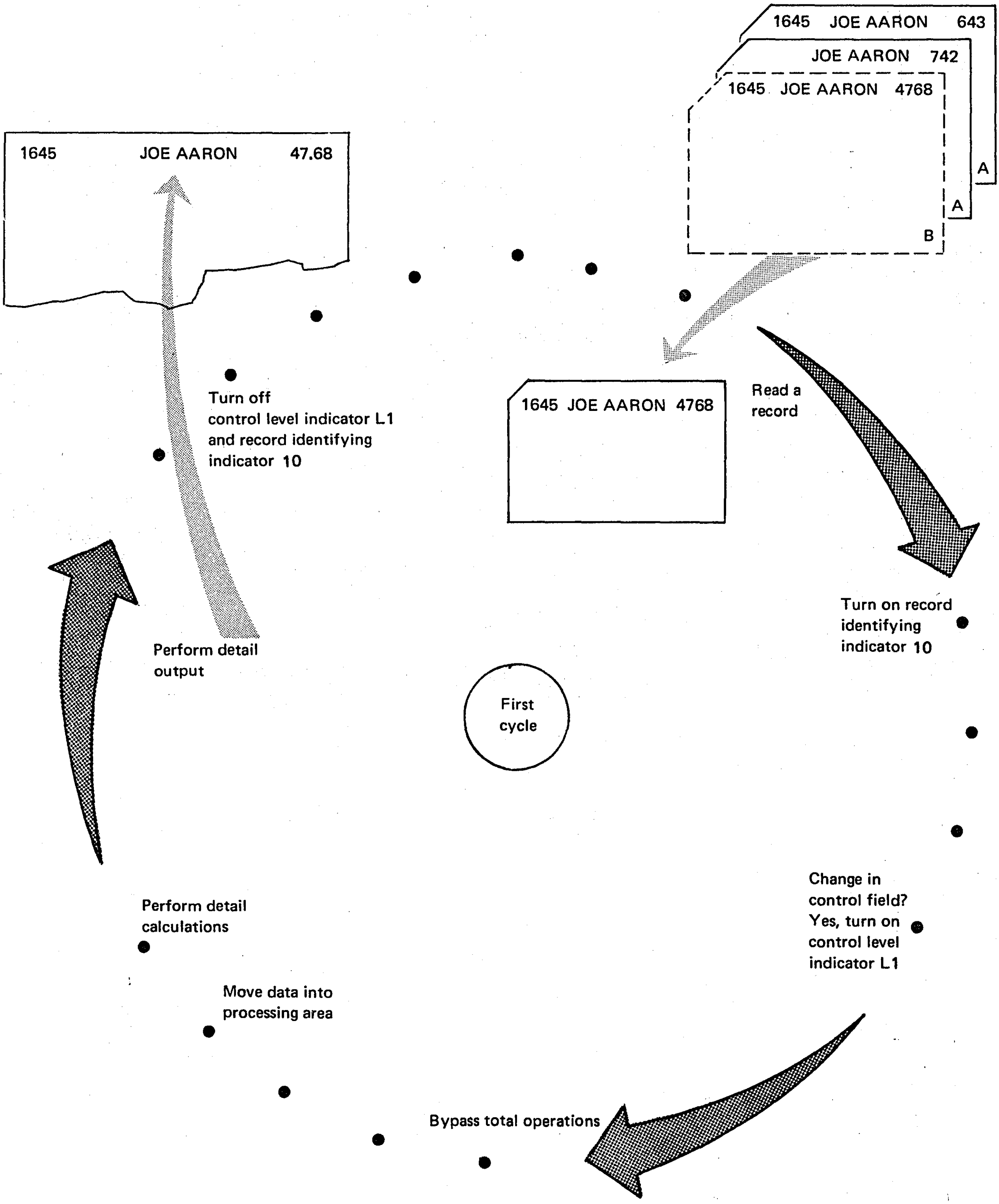

1645 JOE AARON 643
JOE AARON 742
1645 JOE AARON 4768
A
A
B
1645 JOE AARON 47.68
1645 JOE AARON 4768
Read a record
Turn off control level indicator L1 and record identifying indicator 10
Turn on record identifying indicator 10
Perform detail output
First cycle
Change in control field? Yes, turn on control level indicator L1
Perform detail calculations
Move data into processing area
Bypass total operations

Figure 1-26 (Part 1 of 3). Program Cycles Illustrating Use of Halt Indicators

The second cycle shows that operations are performed on the record that contains the blank NUM field. The record containing an amount of 742 has a blank account number field. Thus it is not known whether this record really belongs to Joe Aaron. But Joe is charged 742, regardless, since detail operations are performed before the halt occurs. In order to prevent processing data which could be in error, you must write specifications which will bypass operations when an error occurs. This will be discussed later in the chapter titled *Controlling Operations in an RPG II Program.*

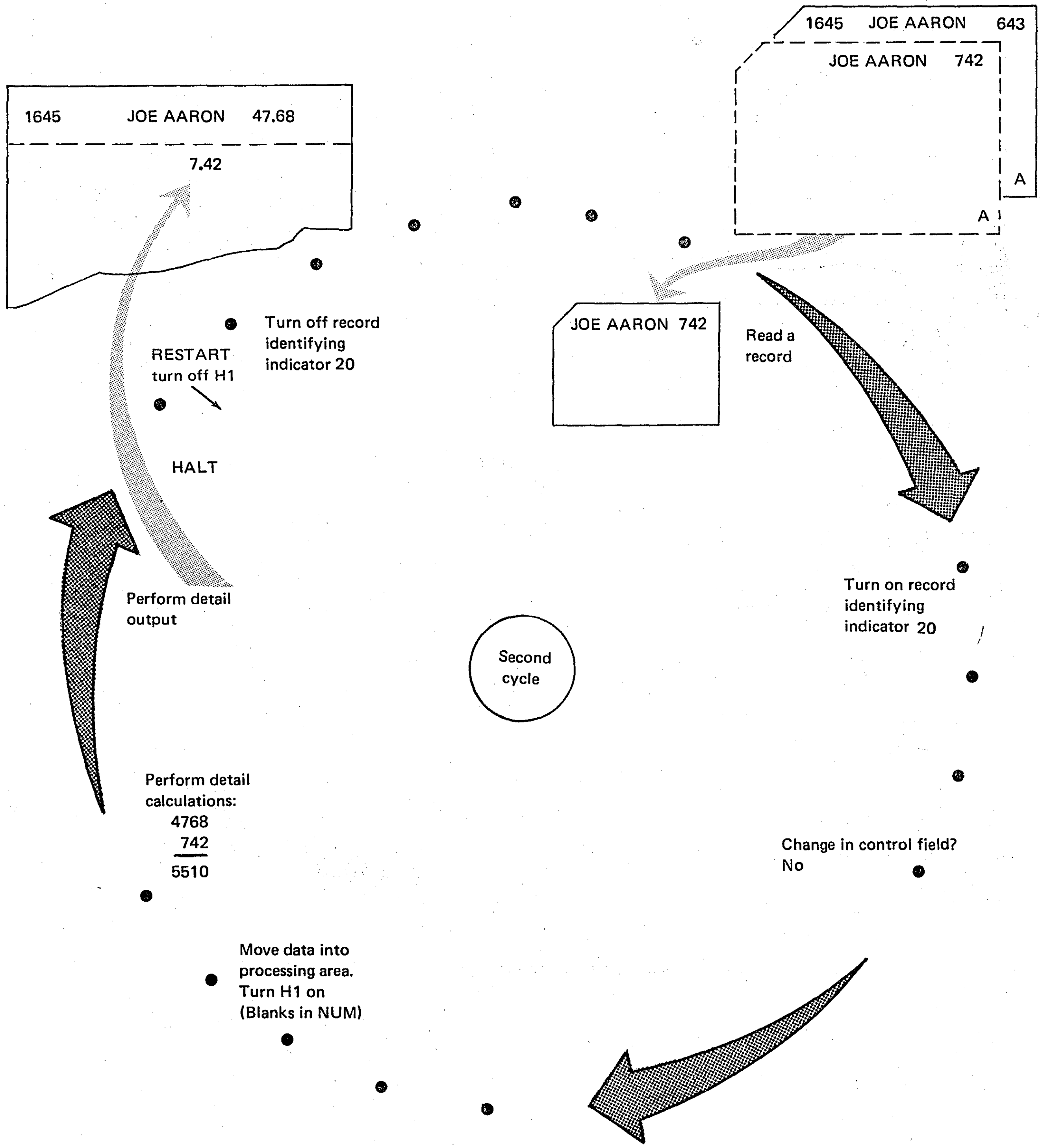

1645 JOE AARON 643
JOE AARON 742
A
A
JOE AARON 742
Read a record
Turn on record identifying indicator 20
Change in control field? No
Second cycle
Move data into processing area. Turn H1 on (Blanks in NUM)
Perform detail calculations:
4768
742
5510
Perform detail output
HALT
RESTART turn off H1
Turn off record identifying indicator 20
1645 JOE AARON 47.68
7.42

Figure 1-26 (Part 2 of 3). Program Cycles Illustrating Use of Halt Indicators

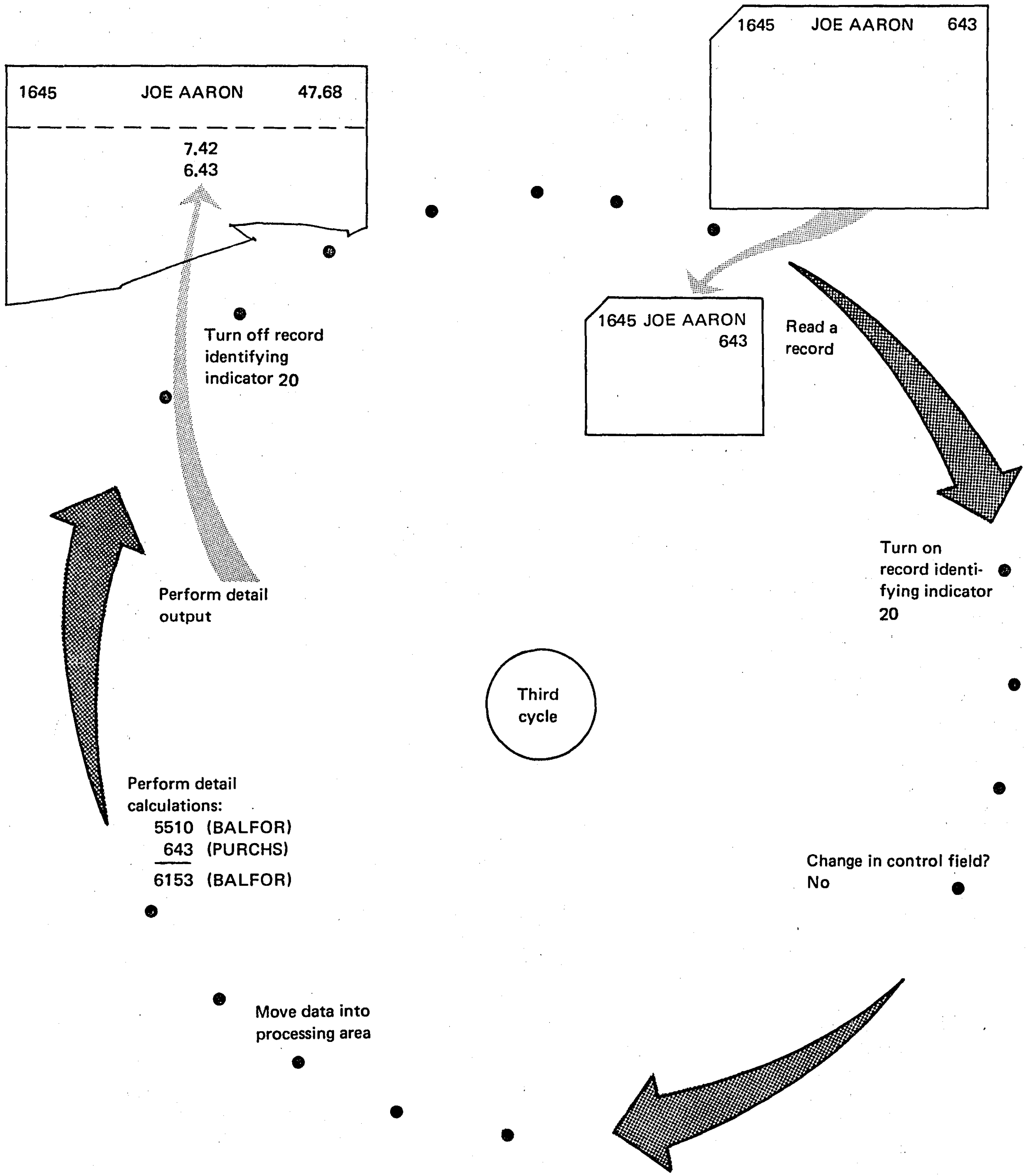
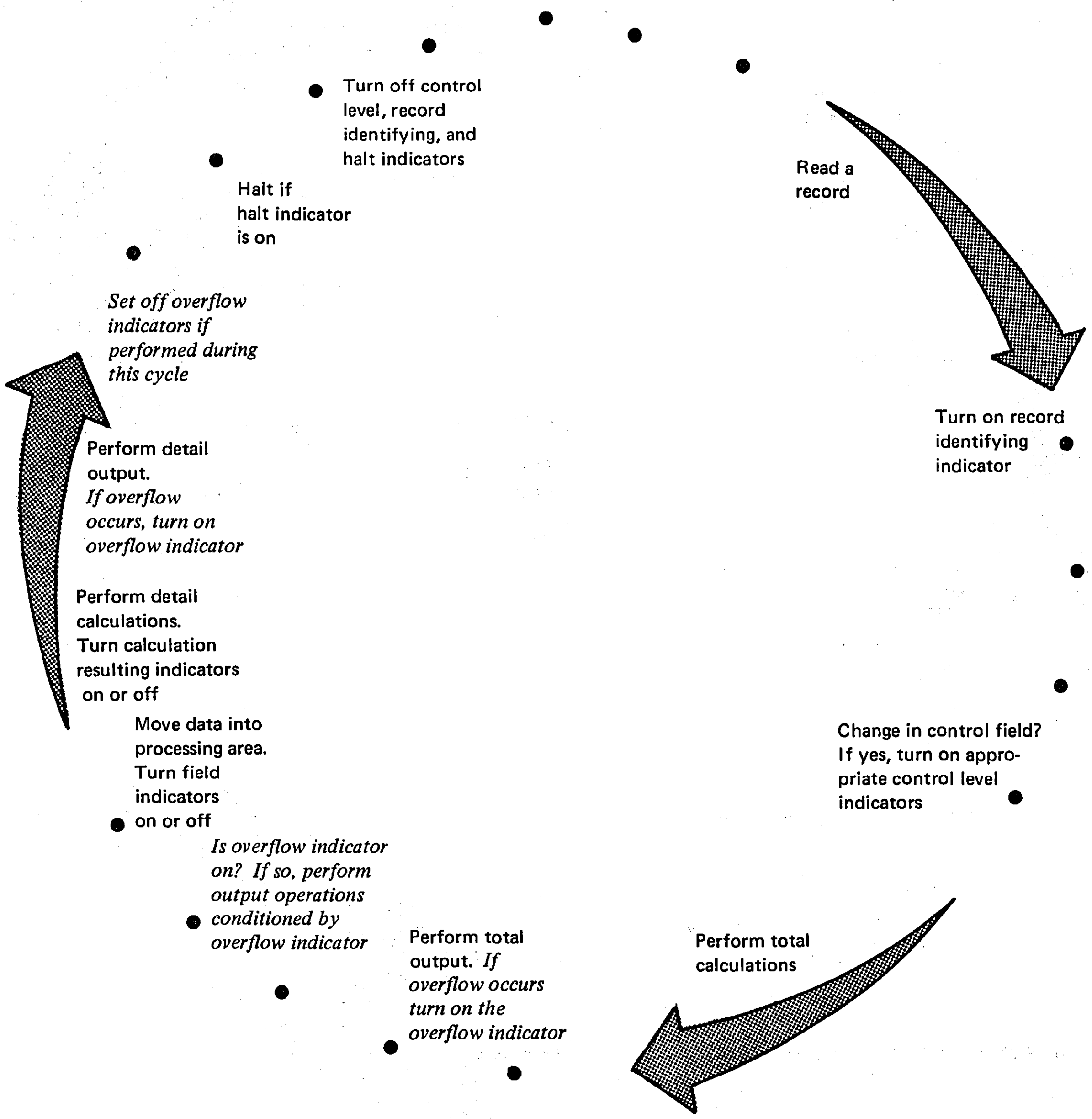

1645 JOE AARON 643
1645 JOE AARON 47.68
7.42
6.43
1645 JOE AARON
643
Read a record
Turn on record identi-fying indicator 20
Change in control field? No
Move data into processing area
Perform detail calculations:
5510 (BALFOR)
643 (PURCHS)
6153 (BALFOR)
Perform detail output
Turn off record identifying indicator 20
Third cycle

Figure 1-26 (Part 3 of 3). Program Cycles Illustrating Use of Halt Indicators

Overflow Indicator (OA-OG, OV)

Overflow indicators are used to signal when the end of a printed page has been reached. They are assigned to the printer file and turn on when the overflow line is printed. This could be either at detail or total output time. Those lines which you wish to print at the end of one page or at the beginning of another are conditioned by the overflow indicator.

Figure 1-27 shows RPG II logic related to overflow indicators. A more detailed discussion of the purpose and use of overflow indicators can be found in the chapter titled *Controlling Printer Output.*



Read a record
Turn on record identifying indicator
Change in control field? If yes, turn on appropriate control level indicators
Perform total calculations
Perform total output. *If overflow occurs turn on the overflow indicator*
Is overflow indicator on? If so, perform output operations conditioned by overflow indicator
Move data into processing area. Turn field indicators on or off
Perform detail calculations. Turn calculation resulting indicators on or off
Perform detail output. *If overflow occurs, turn on overflow indicator*
Set off overflow indicators if performed during this cycle
Halt if halt indicator is on
Turn off control level, record identifying, and halt indicators

Figure 1-27. Logic for Overflow Indicators

Matching Records Indicator (MR)

Thus far, you have been concerned with only one input file. According to RPG II logic discussed so far, a record is read from the input file, then processed. Another is read and processed and so on. Suppose you have more than one input file; from which file is a record read?

RPG II logic has been designed so that your program can select the next record for processing. Figure 1-28 shows general steps in the logic (multifile logic) required when more than one input file is used.

The matching record indicator (MR) is used only when you are processing more than one input file. It indicates when fields on records from different files match. MR is set only after total operations are performed. Thus, at detail time, MR always signals the matching status of the record just selected for processing; at total time, it reflects the matching status of the previous record.

Specific steps in the multi-file logic are described in the chapter titled *Match Fields and Multifile Processing.* At this time, it is sufficient to know at what point in the program cycle records are selected for processing and at what point MR is turned on.

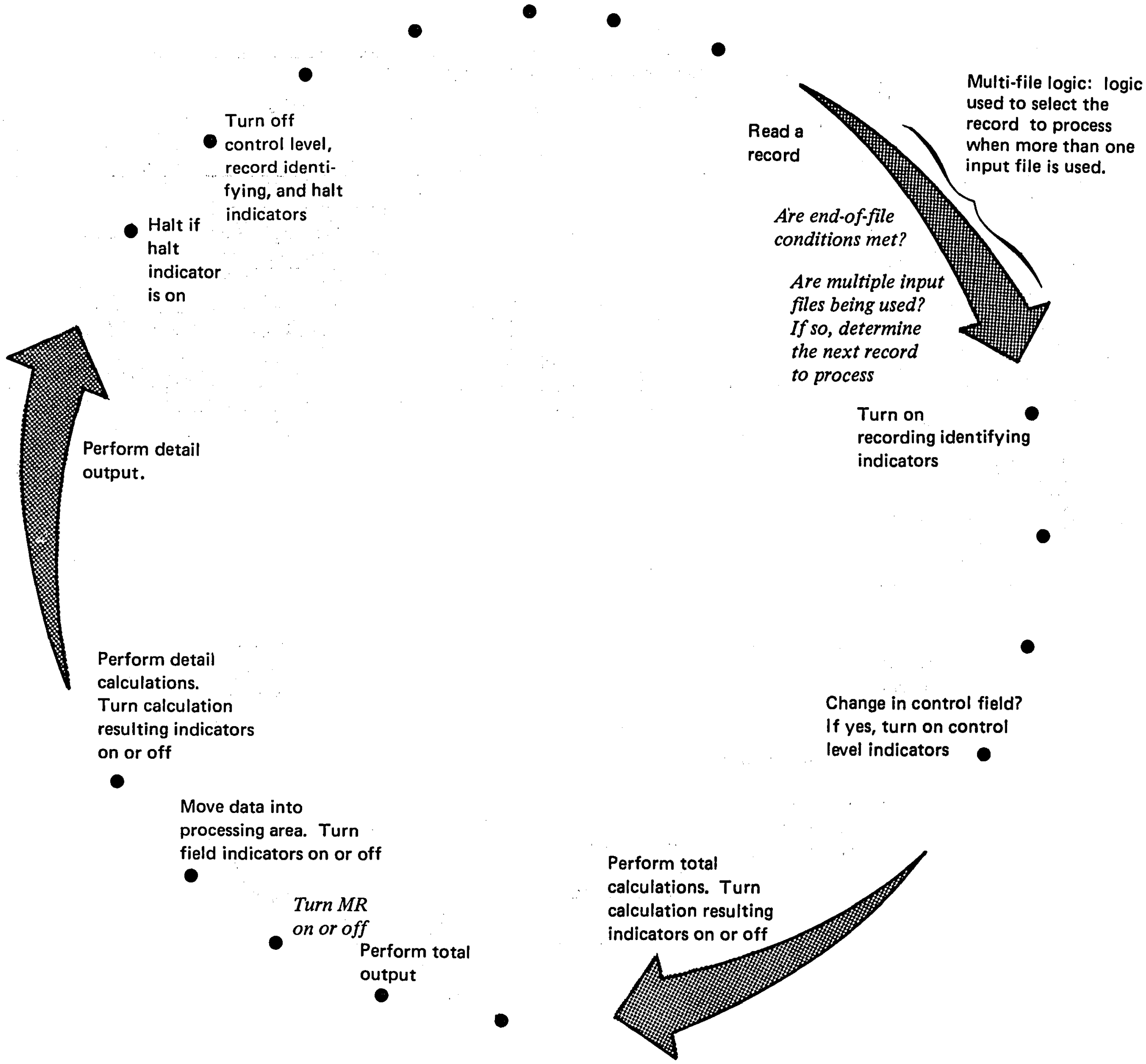
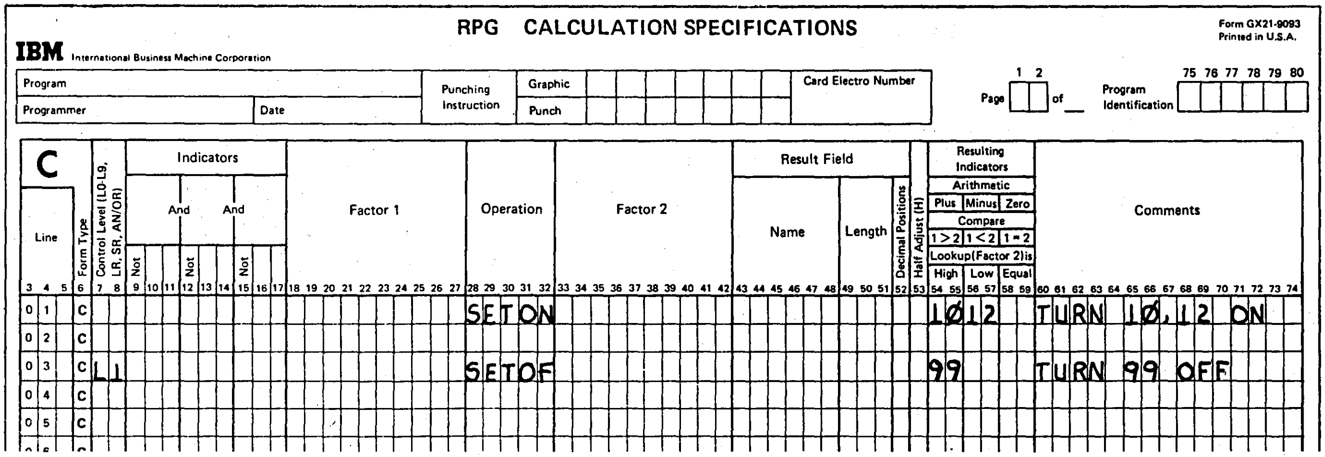

Read a record
Multi-file logic: logic used to select the record to process when more than one input file is used.
Are end-of-file conditions met?
Are multiple input files being used? If so, determine the next record to process
Turn on recording identifying indicators
Change in control field? If yes, turn on control level indicators
Perform total calculations. Turn calculation resulting indicators on or off
Perform total output
Turn MR on or off
Move data into processing area. Turn field indicators on or off
Perform detail calculations. Turn calculation resulting indicators on or off
Perform detail output.
Halt if halt indicator is on
Turn off control level, record identifying, and halt indicators

Figure 1-28. Simplified Matching Record Logic

Setting Indicators

You have just seen the normal setting of indicators according to RPG II logic. You, in your program, can alter this setting by turning any indicator (except 1P) on or off through use of the operation codes SETON and SETOF (see Figure 1-29). An indicator may be set during either detail or total time. It will be set at the time the SETON or SETOF code is executed and will retain the setting you give it until it is reset according to the program logic. (Refer to the logic for the various indicators, earlier in this section, to determine when they are set on and off in the logic cycle.)

Indicators of various types may be used anywhere in the program. For example, you can use LR as a record identifying indicator, L1-L9 as resulting indicators, or L1-L9 as record identifying indicators. If you need to set indicators yourself, you should be thoroughly familiar with RPG II program logic so that you will use the indicators correctly in your program.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page □□ of __ | Program Identification □□□□□□

C

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators (Not, And Not, And Not)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Plus / High / 1>2	Minus / Low / 1<2	Zero / Equal / 1=2	Comments
0 1	C				SETON						1Ø	12		TURN 1Ø,12 ON
0 2	C													
0 3	C	L1			SETOF						99			TURN 99 OFF
0 4	C													
0 5	C													

Figure 1-29. Setting Indicators

Review 1

1. Arrange the following steps in the order they occur in the RPG II logic cycle starting with *Read a record.*
   a. Read a record
   b. Total output
   c. Move data from input area to processing area
   d. Detail calculations
   e. Detail output
   f. Total calculations

2. In the RPG II cycle, total calculations and total output are for data from:
   a. the record just read
   b. records read in previous RPG II cycles

3. When is the 1P indicator on? When is it turned off?

4. Which steps are bypassed during the first program cycle?

5. When the LR indicator comes on, the last program cycle ends after________; therefore________operations are not performed when LR is on (/* record read).

6. When are record identifying indicators turned on? When are they turned off?

7. When are field indicators (the indicators which test the contents of an input field) turned on or off?

8. Halt indicators may be turned on at various times depending on how they are used. If a halt indicator is turned on, when does the computer stop?

9. Calculation resulting indicators are turned on during total or detail calculations. When are they turned off?

Answers To Review 1

1. a, f, b, c, d, e

2. b

3. 1P is on at the beginning of the first program cycle only. It is turned off before the first record is read.

4. total calculations and total output

5. total output, detail

6. Record identifying indicators are turned on right after a record has been read and identified. They are turned off at the end of each RPG II cycle.

7. Field indicators are set just after data is moved from the input area to the processing area.

8. The computer halts after detail output.

9. Resulting indicators remain on until reset by another calculation.

Chapter 2. Describing and Using Input

CHAPTER 2 DESCRIBES:

Specifying and using control fields and split control fields.

Checking the sequence of record types.

Describing input record types using the OR relationship.

OR records with field record relation.

Field record relation with control fields.

Conditioning use of input files.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Function and coding of input fields on the Input sheet.

Function of RPG II indicators.

RPG II object program cycle (Chapter 1).

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Function and RPG II coding for control fields and split control fields.

How to handle typical record type sequence checking situations.

Function and RPG II coding for field record relation.

Uses for conditioning input files.

Setting external indicators.

Note: You can use the review questions contained in *Review 2* at the end of this chapter to test your comprehension of the topics in the chapter. Answers follow the review questions.

INTRODUCTION

For every RPG II program, you must describe the input information you are processing. This includes describing input files, record types within each file, and fields within each record type. Input files are described on the File Description sheet; record types and fields within each input file are described on the Input sheet.

From previous instruction, reading, and experience, you should already know how to describe and use input files, record types, and fields. You should also know how to use RPG II indicators to condition operations. This chapter describes additional ways to use input with control level indicators, field record relation indicators, and external indicators.

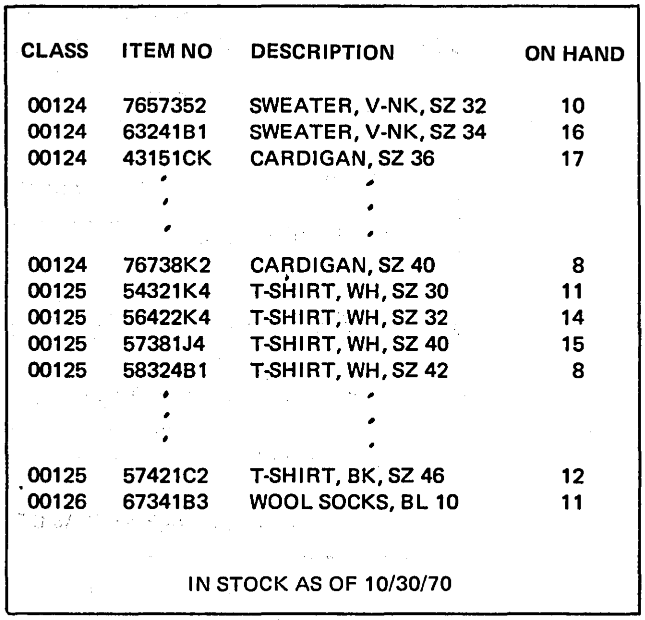
CONTROL FIELDS

A basic type of report in any data processing installation is a detail list that consists of one line of printing for each record read, such as a transaction listing. Figure 2-1 shows what a detail report would look like.

Because product classes are repeated for each line, the report is cluttered and hard to read. The same report (Figure 2-2) grouped by class is much easier to read. Here, all items from one class are listed together with headings used on each page to identify the information. Since all items on one page apply to the same class, the class is printed only once. Such a report is sometimes referred to as a *group-indicated* report. Group-indication is the printing of control information on one line per group. The date is printed at the bottom.

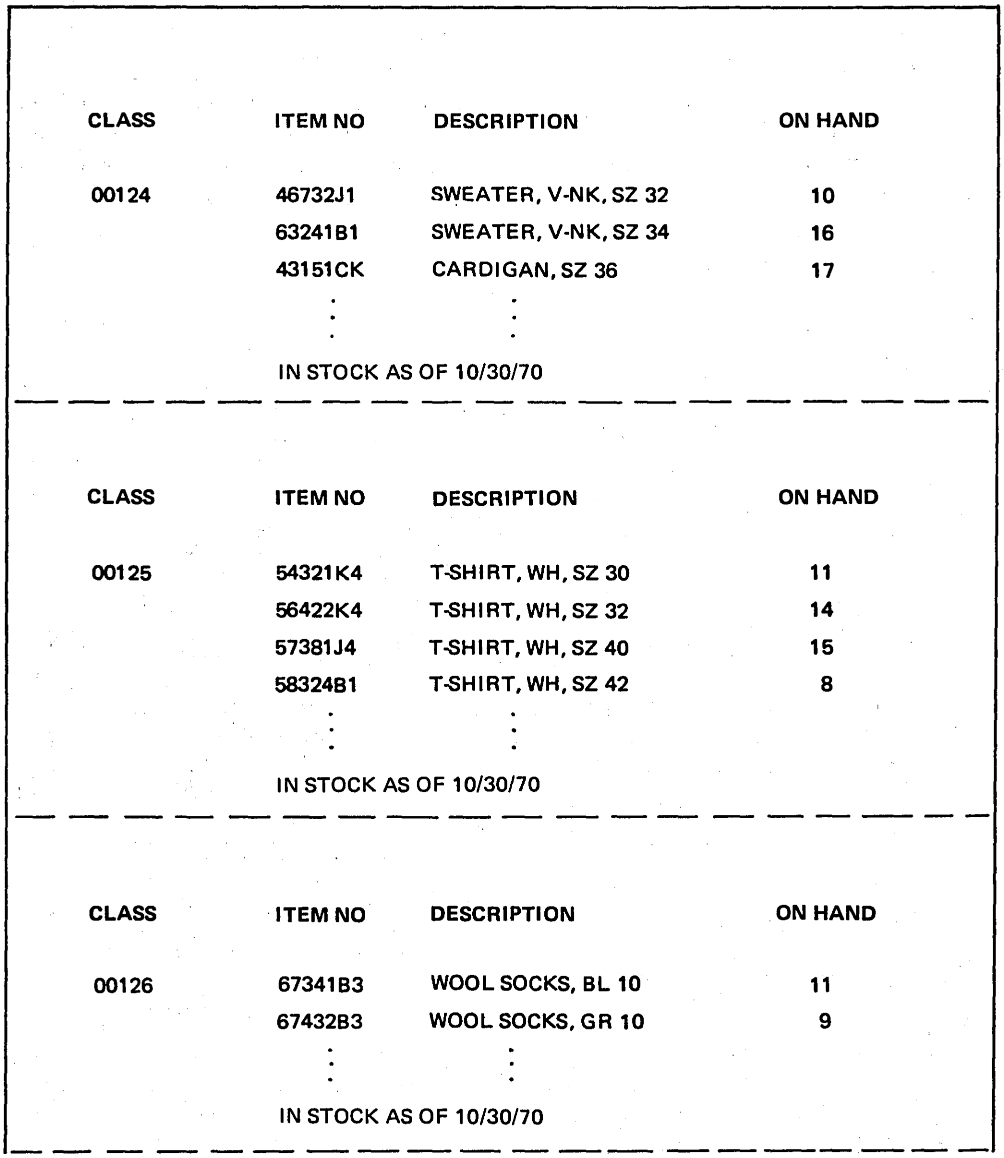
A *control field* is any field used to indicate when a certain type of processing should be done. Since the CLASS field (Figure 2-3) controls processing, it must be specified as the control field. Each time a record is read, this control field is checked for a change in contents (control break). When a control break occurs, a different type of processing or additional processing is to occur. In this case, a change in the CLASS field indicates:

1. Skip to the bottom of the page.
2. Print the date.
3. Skip to a new page.
4. Print heading.



```
CLASS   ITEM NO   DESCRIPTION              ON HAND

00124   7657352   SWEATER, V-NK, SZ 32        10
00124   63241B1   SWEATER, V-NK, SZ 34        16
00124   43151CK   CARDIGAN, SZ 36             17
           .              .
           .              .
           .              .
00124   76738K2   CARDIGAN, SZ 40              8
00125   54321K4   T-SHIRT, WH, SZ 30          11
00125   56422K4   T-SHIRT, WH, SZ 32          14
00125   57381J4   T-SHIRT, WH, SZ 40          15
00125   58324B1   T-SHIRT, WH, SZ 42           8
           .              .
           .              .
           .              .
00125   57421C2   T-SHIRT, BK, SZ 46          12
00126   67341B3   WOOL SOCKS, BL 10           11


              IN STOCK AS OF 10/30/70
```

Figure 2-1. Printed Report of all Items in Stock



```
CLASS          ITEM NO     DESCRIPTION                    ON HAND

00124          46732J1     SWEATER, V-NK, SZ 32              10
               63241B1     SWEATER, V-NK, SZ 34              16
               43151CK     CARDIGAN, SZ 36                   17
                  .             .
                  .             .
                  .             .
               IN STOCK AS OF 10/30/70
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CLASS          ITEM NO     DESCRIPTION                    ON HAND

00125          54321K4     T-SHIRT, WH, SZ 30                11
               56422K4     T-SHIRT, WH, SZ 32                14
               57381J4     T-SHIRT, WH, SZ 40                15
               58324B1     T-SHIRT, WH, SZ 42                 8
                  .             .
                  .             .
                  .             .
               IN STOCK AS OF 10/30/70
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CLASS          ITEM NO     DESCRIPTION                    ON HAND

00126          67341B3     WOOL SOCKS, BL 10                 11
               67432B3     WOOL SOCKS, GR 10                  9
                  .             .
                  .             .
                  .             .
               IN STOCK AS OF 10/30/70
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 2-2. Report Group - Indicated by Product Class

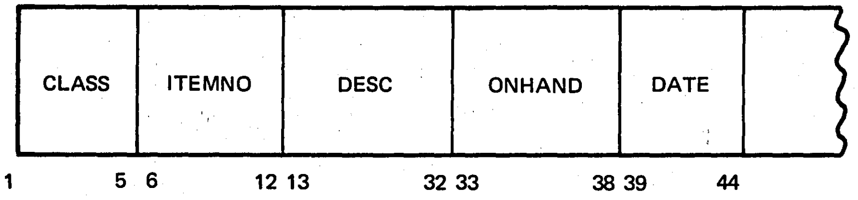
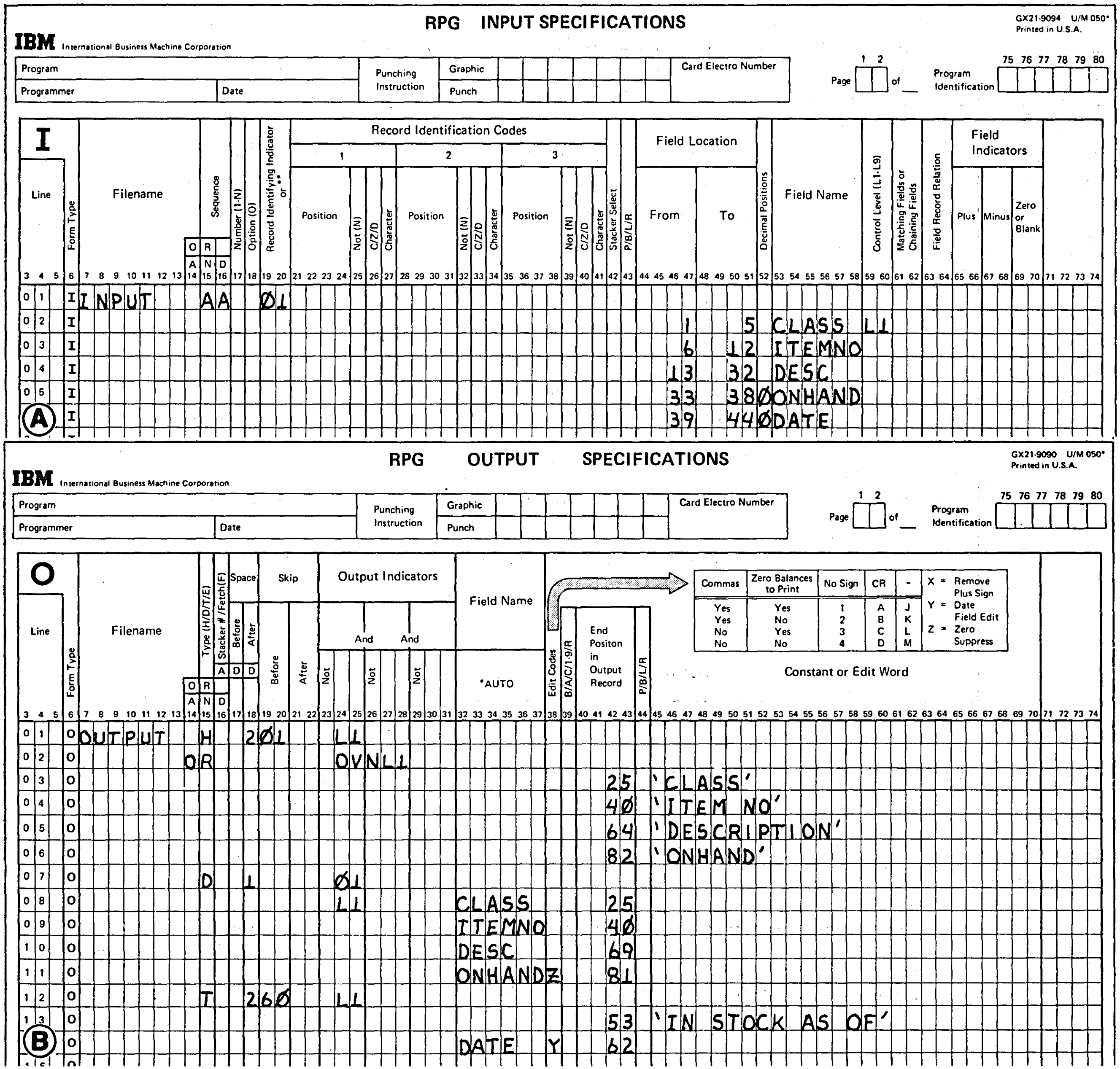


Figure 2-3. Item Record

Coding Control Fields

The RPG II specifications for the program are shown in Figure 2-4. The entry L1 on line 02 of the Input Sheet (Figure 2-4, insert A) establishes the CLASS field as a control field. When the information in the control field changes (a control break occurs) L1 is turned on. The L1 indicator is used on the Output-Format Sheet (Figure 2-4, insert B) to condition those operations which should be performed only when a control break occurs. Note that the L1 indicator is used in line 08 to condition the CLASS field in the detail output line. This causes the CLASS field to be printed only for the first record of a new control group. That is, the CLASS field is printed only when it changes.



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

```
Line Form Filename  Seq  Rec.Id.Ind.   From  To  Dec  Field Name  Control Level
     Type
0 1  I    INPUT     AA   01
0 2  I                                    1    5       CLASS       L1
0 3  I                                    6   12       ITEMNO
0 4  I                                   13   32       DESC
0 5  I                                   33   38  0    ONHAND
(A)  I                                   39   44  0    DATE
```

RPG OUTPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9090 U/M 050* Printed in U.S.A.

```
Line Form Filename  Type Space Skip   Output      Field Name  Edit  End   Constant or Edit Word
     Type                 After Before Indicators              Codes Pos.
0 1  O    OUTPUT    H    2     01     L1
0 2  O            OR                  OVNL1
0 3  O                                                               25   'CLASS'
0 4  O                                                               40   'ITEM NO'
0 5  O                                                               64   'DESCRIPTION'
0 6  O                                                               82   'ONHAND'
0 7  O              D    1            01
0 8  O                                L1          CLASS              25
0 9  O                                            ITEMNO             40
1 0  O                                            DESC               69
1 1  O                                            ONHANDZ            81
1 2  O              T    2     60     L1
1 3  O                                                               53   'IN STOCK AS OF'
(B)  O                                            DATE        Y      62
```

Figure 2-4. Defining and Using a Control Field

Split Control Fields

Two separate parts of a field or two separate fields can be used as *one* control field known as a split control field. This is done by assigning the same control level indicator to both parts of the field. The compiler will consider the data in the split control fields as *one* continuous field.

Suppose you have a 3-character customer number field in the record and now need a 6-character field. The problem is how to put a larger customer number (such as 100010, 100020) in a 3-character field. You cannot change records easily because there is no room for expansion on either side of the customer number field (Figure 2-5), and to expand the field, the entire record format would have to be changed. All programs using these records would also have to be changed to accommodate the changed record format. This would be considerable work and inconvenience. RPG II provides the split control field feature to meet changing data processing needs with minimum effort.

The solution to the problem is to add a 3-character portion to the customer number field using three columns which are not adjacent to the original customer number field (Figure 2-6). The original three numerals of the customer number remain in the original field. The three additional numbers are put in the new customer number field.

At the end of each month, a report is produced consisting of:

1. Customer number.

2. A description of each purchase.

3. The cost of each purchase.

4. The total cost of all purchases.

CUSTNO	ITEMNO	DESC	QTYORD	COST	
1 3	4 12	13 32	33 37	38 44	

Figure 2-5. Three Digit Customer Field

CNUM2	ITEMNO	DESC	QTYORD	COST	CNUM1	
1 3	4 12	13 32	33 37	38 44	45 47	48

Figure 2-6. One Customer Number Split into Two Parts

The report is group-indicated as shown in Figure 2-7.

The customer number determines when totals would be printed and thus must be used as a control field. However, on each record the customer number is split into two parts (two fields). Both must be used in order to get the correct customer number (Figure 2-8).

Coding Split Control Fields

Split control fields must be described in specification lines which follow one another (Figure 2-8, insert A).

CNUM1, the field in columns 45-47 of the record, must be specified on the Input sheet before CNUM2, the field in positions 1-3. This is required because the three digits in CNUM1 are the first three digits of the customer number.

Parts of a split control field may be either alphameric or numeric. In this example, they were both defined as numeric (indicated by the entry in column 52). If one of them, however, had been defined as numeric and one as alphameric, they both are considered numeric by the compiler.

CHECKING THE SEQUENCE OF RECORD TYPES

Many data processing jobs require the use of several kinds of information. Sometimes, this information must be in a special order to produce the correct results.

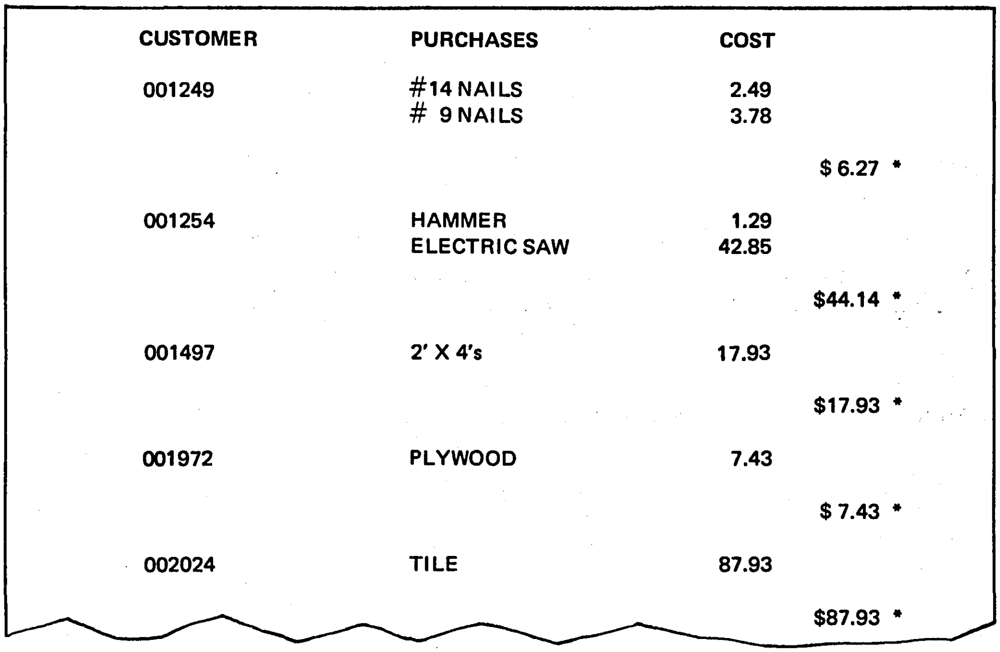
Order of Record Types Within a Group

For example, to do end-of-the-month billing, you need several kinds of information. For each account you must know:

1. The balance forward at the beginning of the month.
2. Payments made during the month.
3. Purchases made during the month.

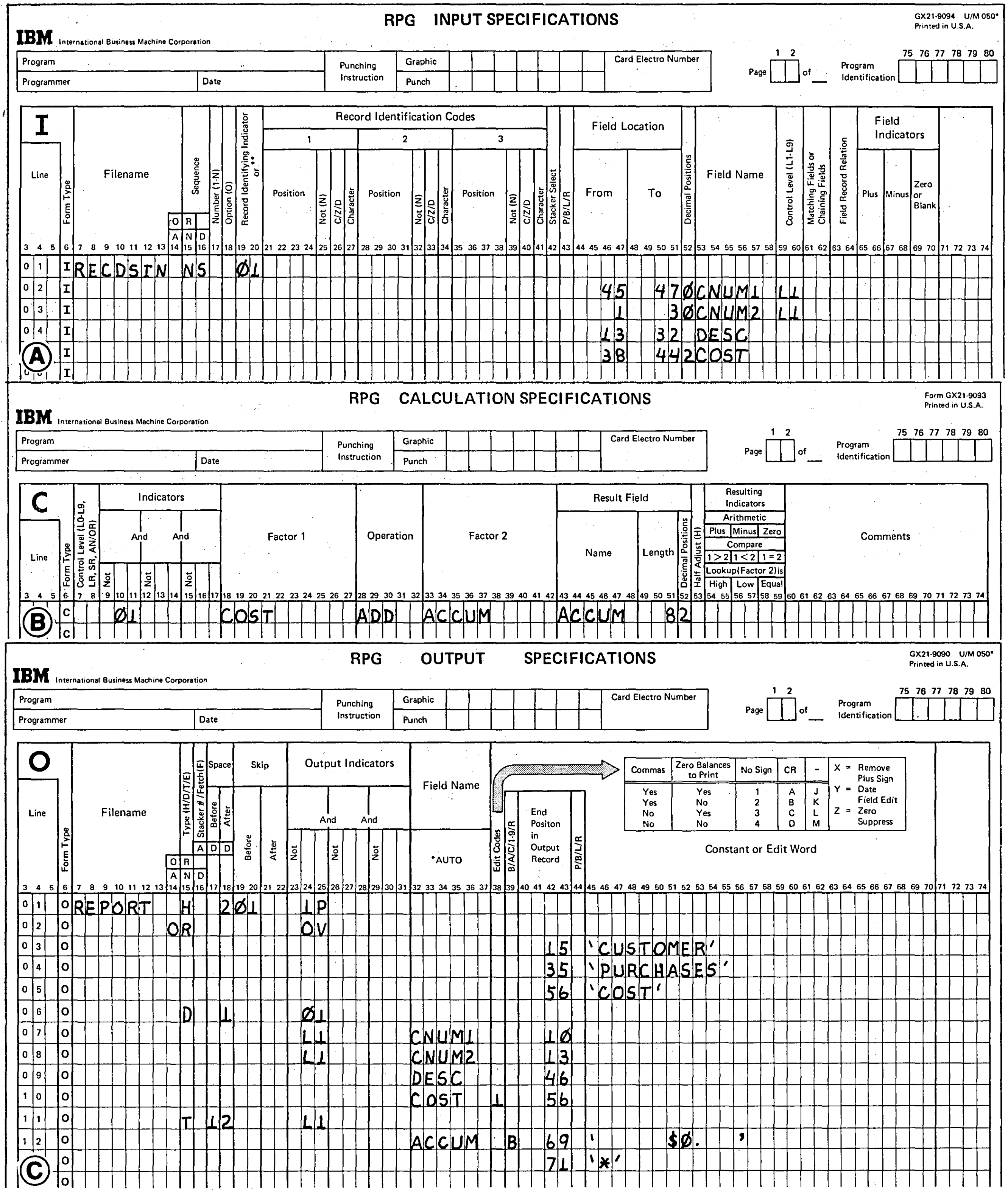
To get the amount due, you subtract payments made from the balance forward and then add new purchases to that amount.

Information concerning balance forward, payments, and purchases is usually on more than one record. Payments are usually recorded as they are received. Purchases are recorded as they are made. The balance forward is also kept on a separate record.



```
CUSTOMER        PURCHASES         COST

001249          #14 NAILS          2.49
                # 9 NAILS          3.78

                                          $ 6.27 *

001254          HAMMER             1.29
                ELECTRIC SAW      42.85

                                          $44.14 *

001497          2' X 4's          17.93

                                          $17.93 *

001972          PLYWOOD            7.43

                                          $ 7.43 *

002024          TILE              87.93

                                          $87.93 *
```

Figure 2-7. Report Group Printed by Customer Number



RPG INPUT SPECIFICATIONS

Line	Form Type	Filename	Sequence	Number	Option	Record Identifying Indicator	From	To	Decimal Positions	Field Name	Control Level
01	I	RECDSIN	NS			01					
02	I						45	47	0	CNUM1	L1
03	I						1	3	0	CNUM2	L1
04	I						13	32		DESC	
Ⓐ	I						38	44	2	COST	
	I										

RPG CALCULATION SPECIFICATIONS

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions
Ⓑ	C	01	COST	ADD	ACCUM	ACCUM	8	2
	C							

RPG OUTPUT SPECIFICATIONS

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	Blank After	End Position in Output Record	Constant or Edit Word
01	O	REPORT	H		2	01	1P					
02	O	OR					OV					
03	O										15	'CUSTOMER'
04	O										35	'PURCHASES'
05	O										56	'COST'
06	O		D		1		01					
07	O						L1	CNUM1			10	
08	O						L1	CNUM2			13	
09	O							DESC			46	
10	O							COST	1		56	
11	O		T	1	2		L1					
12	O							ACCUM		B	69	' $0. '
Ⓒ	O										71	'*'
	O											

Figure 2-8. Using a Split Control Field

Thus, to do the billing, three different types of records are necessary for each account. Furthermore, these records must be read in a special order. The balance forward record must come first. The payment and purchase records can be in any order; it makes no difference whether you subtract all payments first, or add all receipts first, or add receipts and subtract payments in any order. However, you must decide on the order of these records for your program and keep them in that order, since the computer will always expect them in a certain order.

Management of a retail store requires all receipts to be listed and subtracted before purchases are listed and added. Thus, the order in which the records must be read is:

1. Balance Forward.

2. Payments.

3. Purchases.

Remember that these three types of records are necessary for *one* account. When they are organized, they are organized according to account. Each record has a name on it. All records with the same name must be grouped together in the file and must be in the order indicated (see Figure 2-9, part A).

What if one of the records accidentally got out of order? Some customer would end up with the wrong amount.

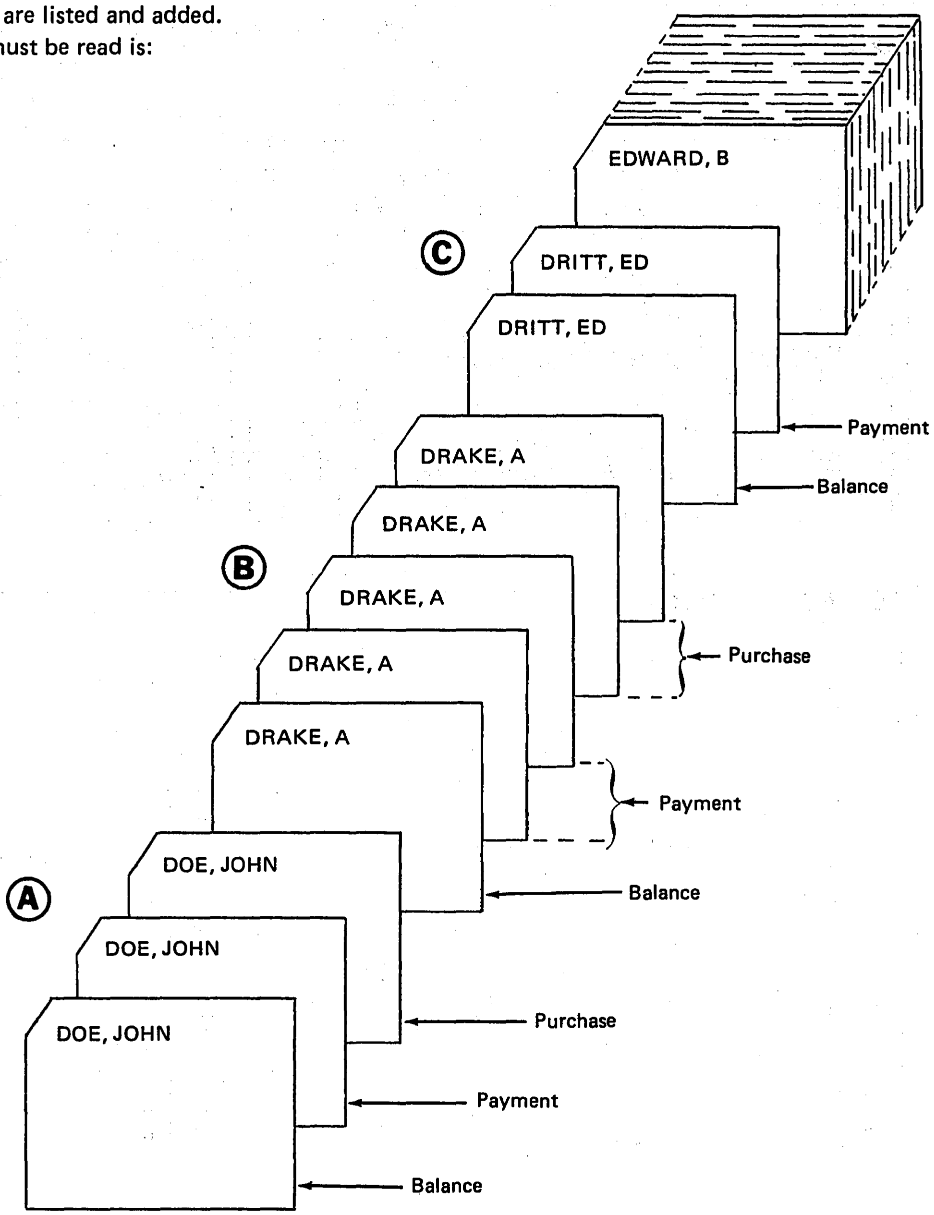
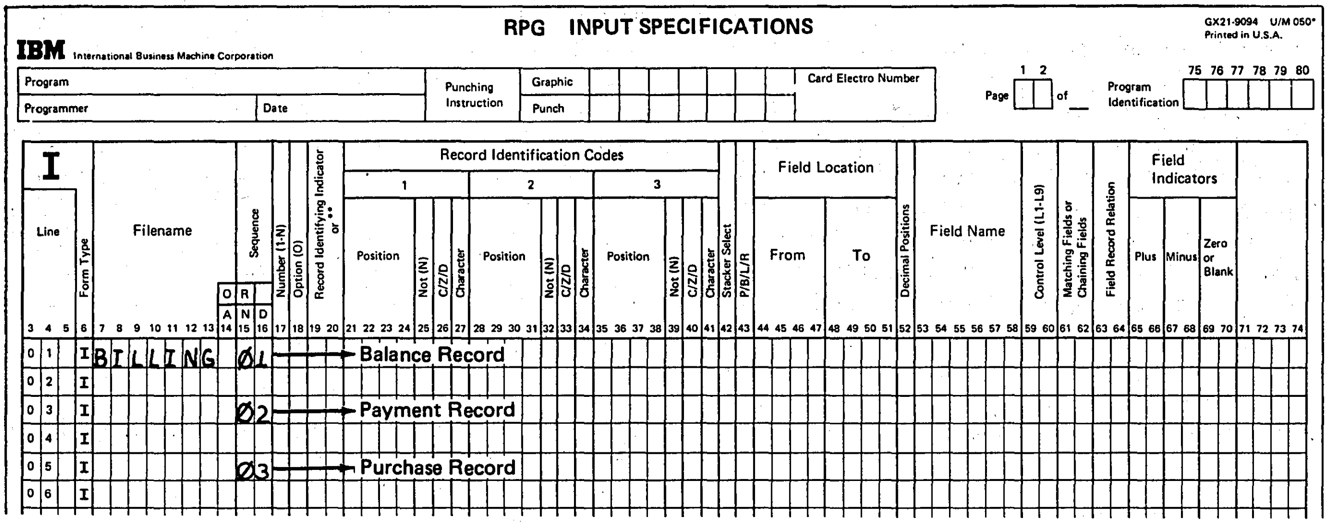


Figure 2-9. Order of Record Types Within a Group

To prevent this, you direct your program to check for record sequence by entries in columns 15-16 on the Input sheet. These columns are titled *Sequence* and are used for indicating the sequence of record types for each account.

These sequence columns are used for *every* program you write. If you are not checking for a special order, these columns must contain alphabetic entries. If you are checking for a special order, these columns must contain numeric entries (01-99).

Since it is first, the Balance record must be given the sequence entry 01 (see Figure 2-10). What sequence entries would the Payment and Purchase records have? Logically, they would be 02 and 03 respectively (Figure 2-10). However, you could have used 09 and 20. You may use any numbers from 01-99 just as long as the numbers used are in ascending order. However, 01 must always be used.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page □□ of __ | Program Identification

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	
01	I	BILLING	Ø1	→ Balance Record
02	I			
03	I		Ø2	→ Payment Record
04	I			
05	I		Ø3	→ Purchase Record
06	I			

Figure 2-10. Record Sequence

Since there are three different types of records, each should be identified by a code so that the computer knows which record type was read. Record identifying indicators should be specified for each record type. All fields in each record type must also be described. Figure 2-11 shows how the records kept by the retail store were described.

How does the entry specifying sequence help you check for record sequence? Suppose a payment record was read after a purchase record. This would be incorrect order. The program knows that a 02 record type doesn't come after a 03 type and will automatically halt because of the mistake.

More Than One Record Type Per Group

What would happen if two payment records (02 record type) were read in a row? The program would halt because it expects a 03 record type to follow a 02 type. It does not expect two 02 types in a row. But what if a customer actually made two payments during the month? Or what if he bought more than one item during the month? You wouldn't want the program to halt whenever it read more than one payment of purchase record per customer.

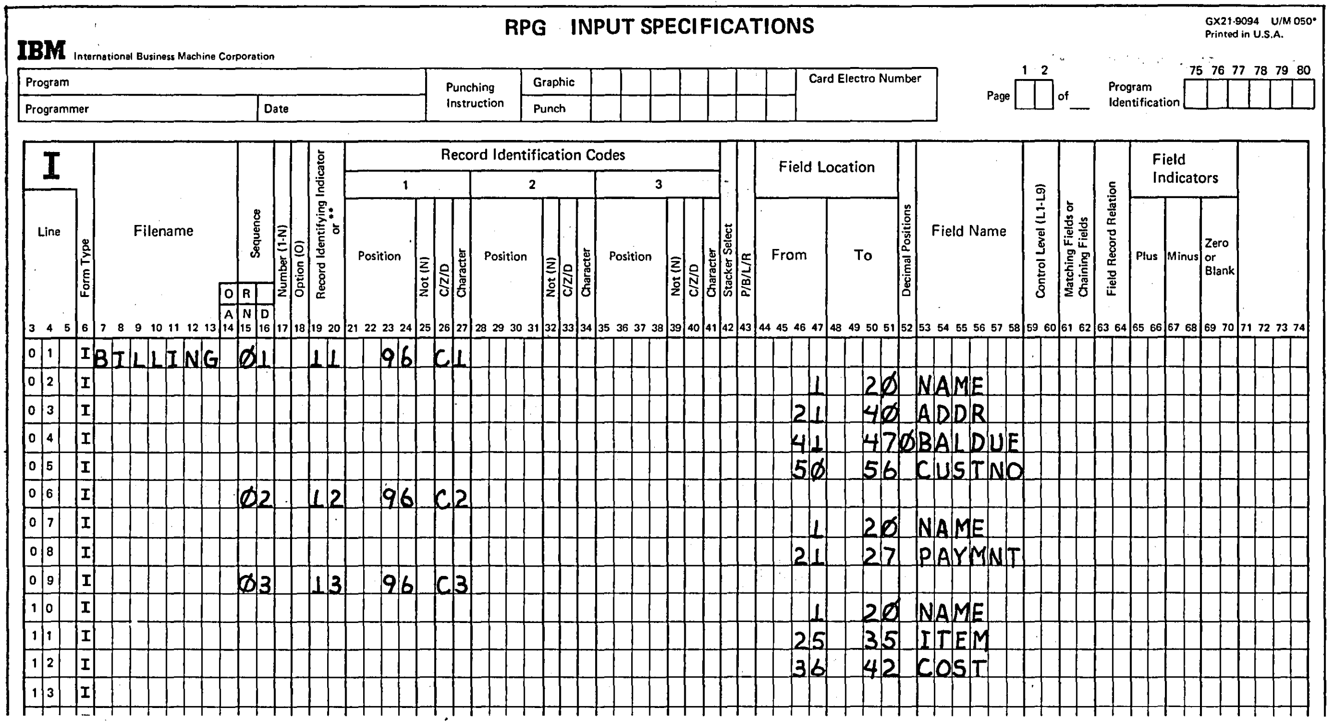
You must make another entry to indicate whether the program can expect to read one or more records of the same type in one group. This entry is made in column 17 (Number) of the Input sheet. A 1 indicates only one record per type; an N indicates one or more records per type. In this example, only one balance record is needed. However, there may be more than one payment record or purchase record. Figure 2-12 shows these entries.

Optional Record Types in the Group

It is also possible in this example to have no purchase record. A customer might not have purchased anything during the month. If so, a balance record would be read after a payment record for the previous customer (see Figure 2-9, part C). According to the specifications shown, this is incorrect order. The program would halt.

To prevent halting in this situation, you must make another entry, this time in column 18. Place the letter O in column 18 to indicate that the record type is optional (it may or may not be present). If you leave column 18 blank, the computer assumes that the record type *must always* be present.

Of the three record types, which must always be there? The balance forward record should be present. Leave column 18 blank for it. The other two record types are optional. Enter the letter O for each (see Figure 2-13).



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

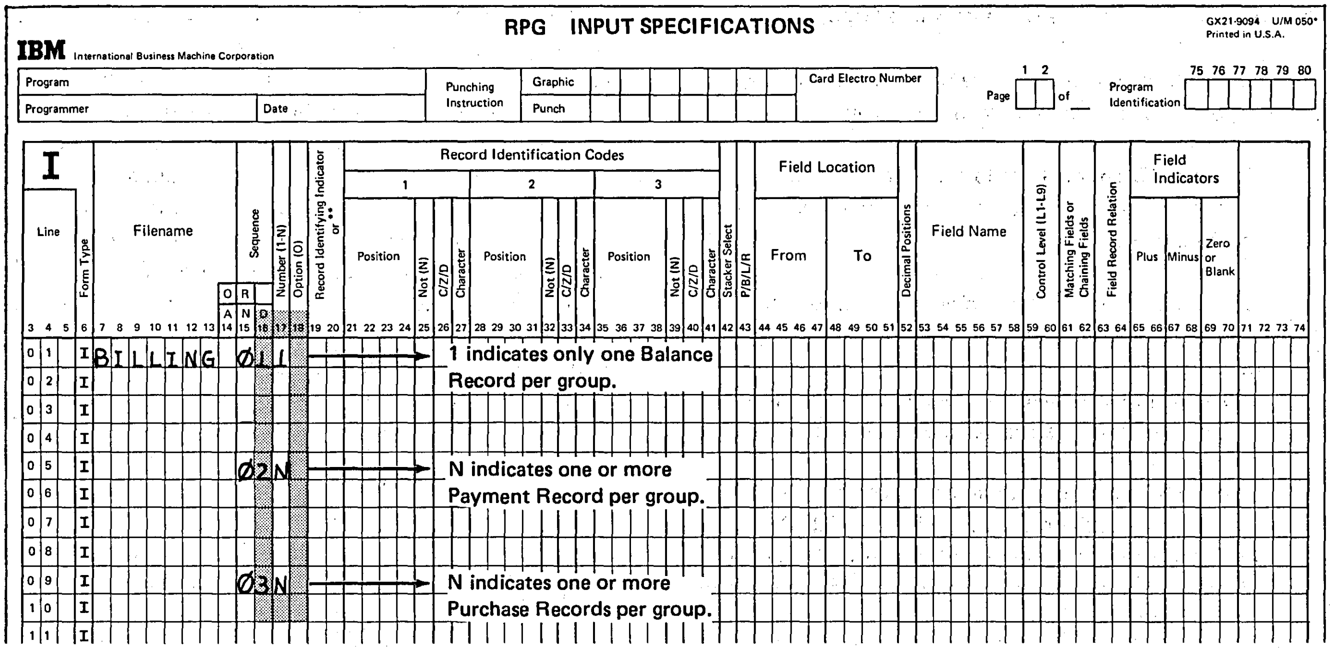
GX21-9094 U/M 050*
Printed in U.S.A.

Program | Punching Instruction: Graphic / Punch | Card Electro Number | Page □□ of ___ | Program Identification (75 76 77 78 79 80)

Programmer | Date

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position (1)	Not (N)	C/Z/D	Character	Position (2)	Not (N)	C/Z/D	Character	Position (3)	Not (N)	C/Z/D	Character	Stacker Select	P/B/L/R	From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Plus	Minus	Zero or Blank	
01	I	BILLING	Ø1			11	96		C	1																					
02	I																				1	2Ø		NAME							
03	I																				21	4Ø		ADDR							
04	I																				41	47	Ø	BALDUE							
05	I																				5Ø	56		CUSTNO							
06	I		Ø2			12	96		C	2																					
07	I																				1	2Ø		NAME							
08	I																				21	27		PAYMNT							
09	I		Ø3			13	96		C	3																					
10	I																				1	2Ø		NAME							
11	I																				25	35		ITEM							
12	I																				36	42		COST							
13	I																														

Figure 2-11. Description of Sequenced Record Types



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

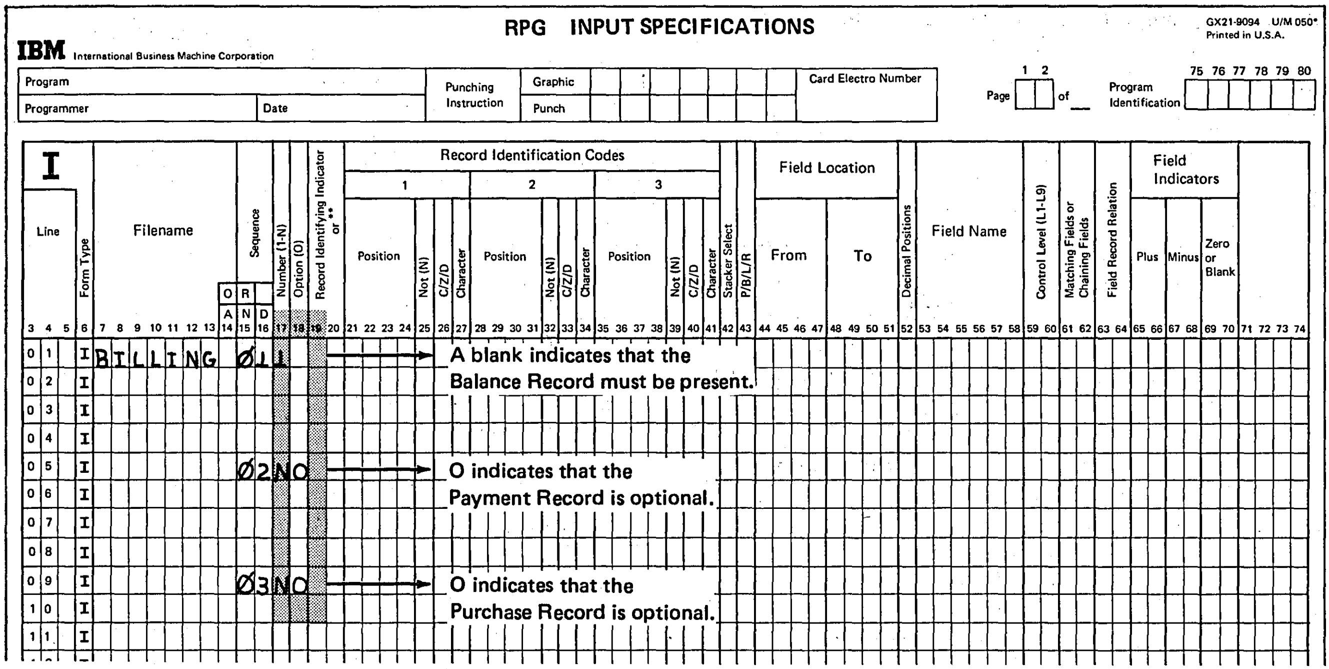
```
Line  Form  Filename  Seq  Number  Option
       Type          (15-16) (1-N)  (O)
01     I    BILLING   Ø1    1
02     I
03     I
04     I
05     I              Ø2    N
06     I
07     I
08     I
09     I              Ø3    N
10     I
11     I
```

1 indicates only one Balance Record per group.

N indicates one or more Payment Record per group.

N indicates one or more Purchase Records per group.

Figure 2-12. Number of Each Record Type Per Group



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

```
Line  Form  Filename  Seq  Number  Option
       Type          (15-16) (1-N)  (O)
01     I    BILLING   Ø1    1
02     I
03     I
04     I
05     I              Ø2    N       O
06     I
07     I
08     I
09     I              Ø3    N       O
10     I
11     I
```

A blank indicates that the Balance Record must be present.

O indicates that the Payment Record is optional.

O indicates that the Purchase Record is optional.

Figure 2-13. Optional Record Types in a Group

Checking the Order of Record Types in a Group

In summary, three entries must be made to ensure proper checking of the sequence of records in a group.

1. Columns 15-16 must contain a numeral from 01 through 99 that indicates the order in which records must be read.
2. Column 17 must contain an entry that indicates whether or not more than one record of a type can be expected. A 1 indicates that only one record of a type will be accepted. An N indicates that more than one record of a type will be accepted.
3. Column 18 must contain an entry which indicates whether or not the record type is optional. The letter O indicates that the record type is optional. A blank indicates that the record type must be present.

Incorrect Records Within a Group

The entries for checking the sequence of record types within a group will determine that the records in groups A and B shown in Figure 2-14 are in order. Suppose, however, that the payment records for John Hill and A. James were mixed up (Figure 2-15). The program using the sequence specifications just described would not find this error. The record types are still in proper order, but the records themselves are in the wrong groups. To ensure that records are in the right group other specifications have to be made.

For the end-of-month billing example, the NAME field is used as a control field. A change in the NAME field indicates the end of one group and the start of another. Since the balance record is always the first one in a new group, the balance record type should be the only one that causes a control break. If the records are mixed up, as shown in Figure 2-15, a control break will occur before all records of one group have been read. For example, when the Arnold James' payment record is read after a John Hill balance record, a control break occurs because information in the NAME field changes. There should be no control break at this time. If there is a control break *here,* the results of the report will be inaccurate.

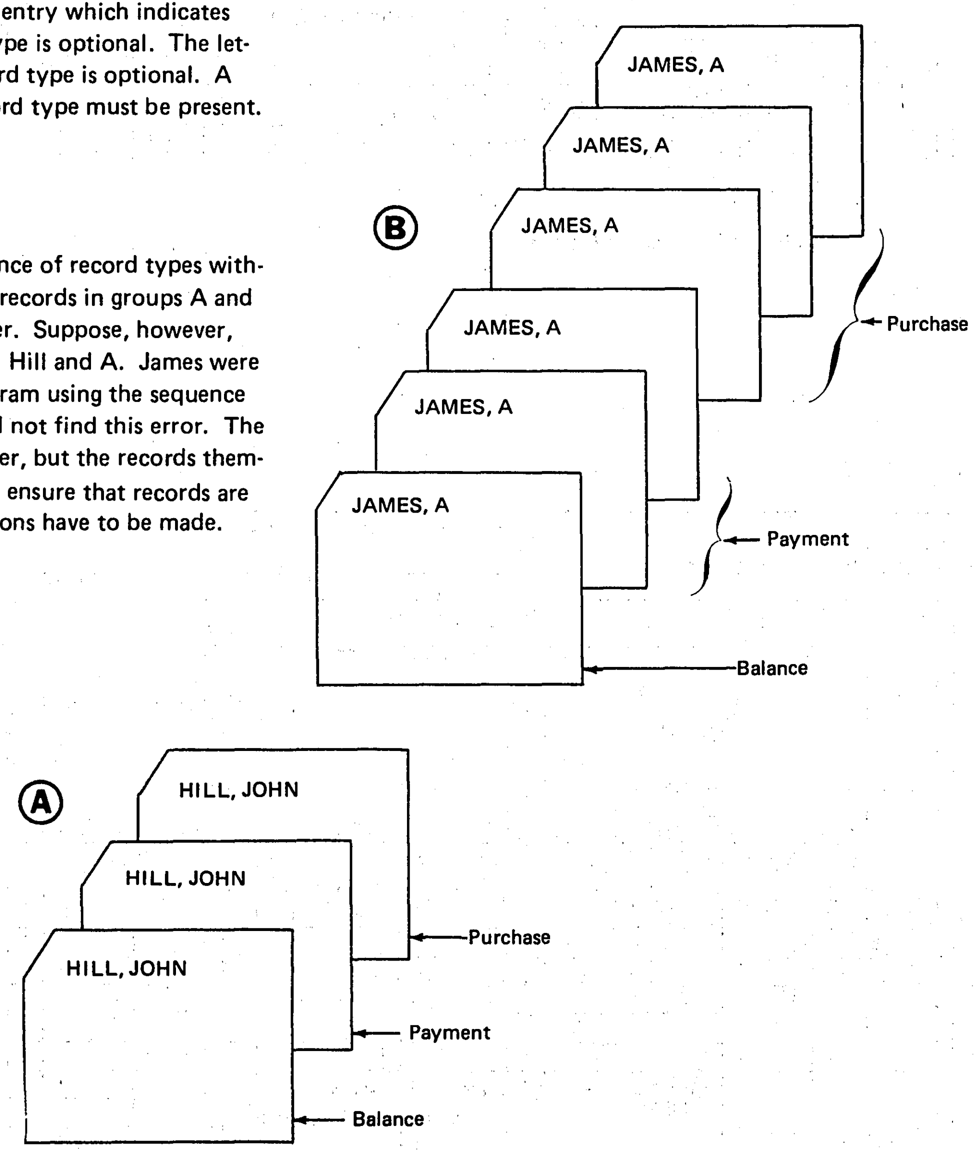


Figure 2-14. Correct Data Records in a Group

To prevent this, certain calculation specifications must be made. Line 01 of the Calculation sheet shown in Figure 2-16 shows that indicator H1 is set on. H1 is a halt indicator. When it is on, processing halts after calculations and output operations have been performed for the record just read.

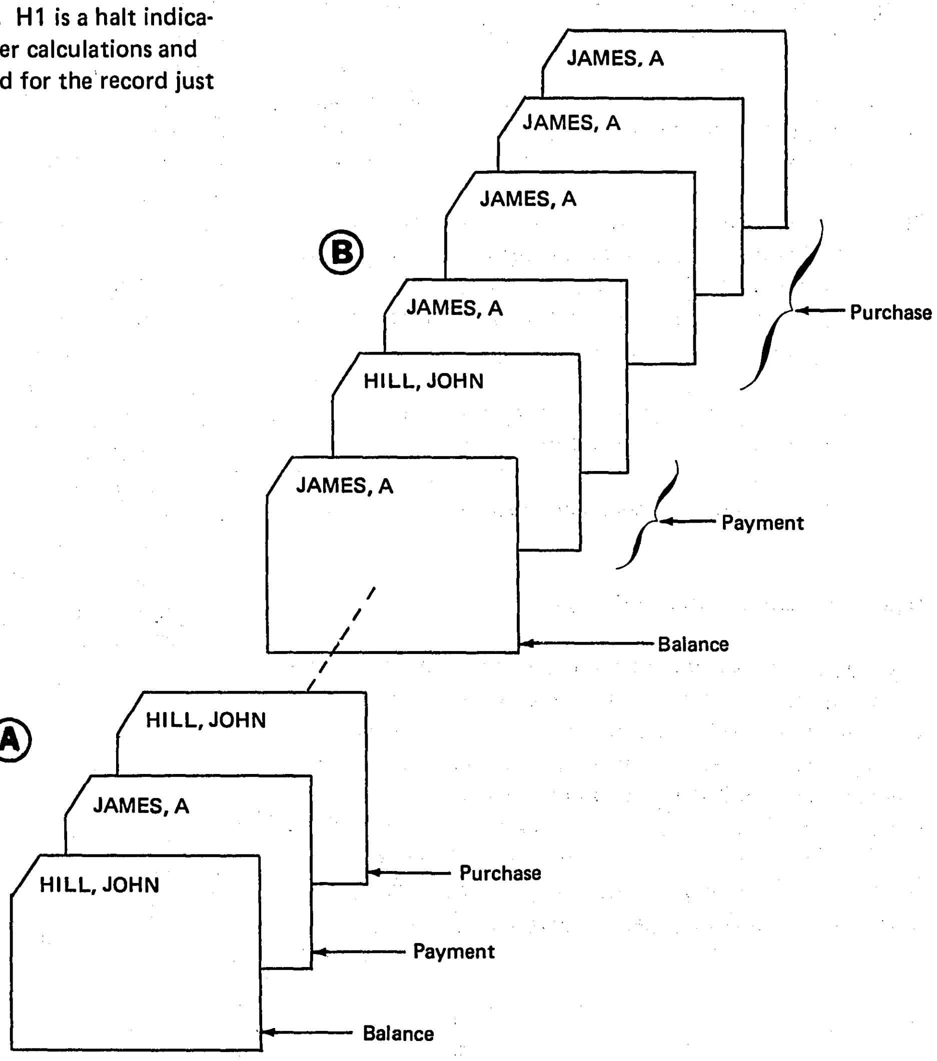
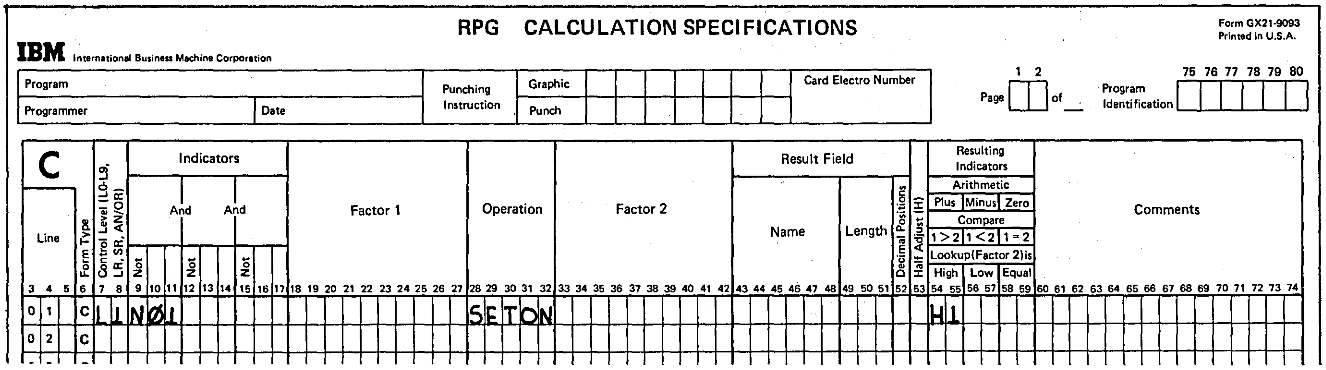


Figure 2-15. Incorrect Data Records in a Group



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer | Date
Punching Instruction | Graphic | Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators (9-11)	And (12-14)	And (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators High (54-55)	Low (56-57)	Equal (58-59)	Comments (60-74)
01	C	L1	NØ1				SETON						H1			
02	C															

Figure 2-16. Haltinყ When Incorrect Record if Found in a Group

Look at the indicators which condition the SETON operation, L1 and N01. A control break (L1 turns on) caused by record type 01 (balance record) is correct. But a control break (L1) caused when any other record type (02 or 03) is read indicates that a record is in the wrong group. This is an error condition. Thus when L1 is on (a control break has occurred) with any record identifying indicator other than 01 (N01), the halt indicator H1 is set on to stop processing.

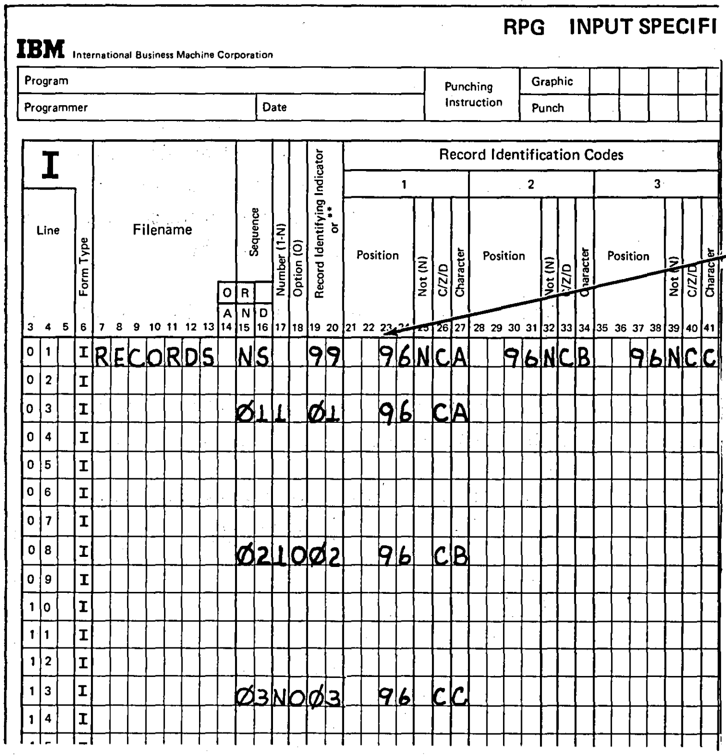
Halting on an error is one way of handling error conditions. This method allows you to stop, correct the record in error, and start processing over again. This often wastes time since you must restart the computer each time an error is found. Programming to bypass the error is more often done. This will be discussed at a later time.

Sequenced and Unsequenced Record Types in a Group

So far we have talked about having records in a group which must be in a special sequence. However, you may also have records in the group which need not be in any sequence. In this case, all records which do not need to be in sequence are specified on the Input sheet before those that do. Remember that unsequenced records must have alphabetic entries in columns 15-16, and blanks in columns 17-18.

Unexpected or Unused Records Within a Group

If the computer reads any record types which are not specified, it will halt. Often you may have several record types within a file, but the job being done requires the use of only a few of the record types. Do you still have to specify each type? No, you don't. But remember each time an undescribed type is found, the program halts. This could result in wasted time. Therefore, to prevent halting and to eliminate a description of each record type, you specify a *catch-all* indicator in addition to specifying all record types needed (see Figure 2-17).



RPG INPUT SPECIFI

IBM International Business Machine Corporation

Program

Programmer

Date

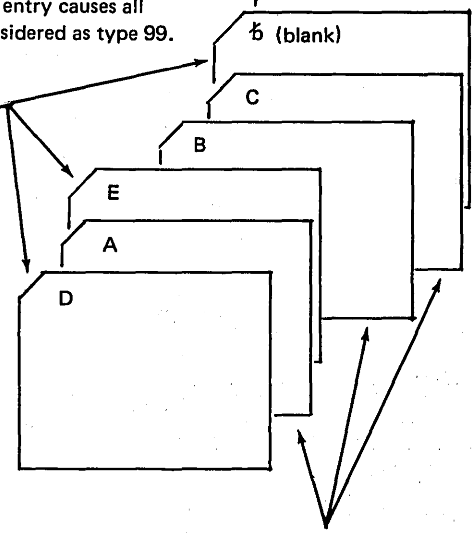
Punching Instruction

Graphic

Punch

I

							Record Identification Codes 1				Record Identification Codes 2				Record Identification Codes 3			
Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character
01	I	RECORDS	NS			99	96	N	C	A	96	N	C	B	96	N	C	C
02	I																	
03	I		Ø1	1		Ø1	96		C	A								
04	I																	
05	I																	
06	I																	
07	I																	
08	I		Ø2	1	O	Ø2	96		C	B								
09	I																	
10	I																	
11	I																	
12	I																	
13	I		Ø3	N	O	Ø3	96		C	C								
14	I																	



These record types are not described for they are not needed. The catch-all sequence entry causes all to be considered as type 99.

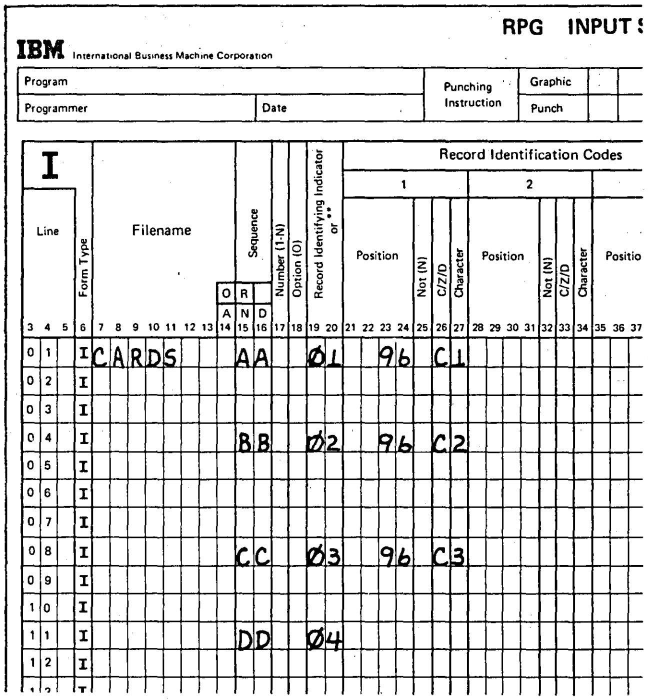
These record types are individually described because they are used for the job.

Figure 2-17. Catch-All Sequence Entry

According to the specifications in Figure 2-17, any record read which does *not* have one of the identification codes specified, is considered to be record type 99. If no operations are conditioned by record identifying indicator 99, none will be done for all records which are considered type 99.

You may also use a catch-all indicator specification to prevent halting when unexpected records (record in wrong file, blank record, etc.) are read. Unwanted card records are normally stacker selected into a special stacker so that they can be removed from the deck at the end of the job.

Figure 2-18 shows specifications that describe three unsequenced record types used in the program and a catch-all indicator which will be assigned to unwanted record types found in the file. When records are not in a special order, (alphabetic entries in columns 15-16), the catch-all indicator is described last with no Record Identification Codes. The catch-all indicator turns on if a card is read which can not be identified by any of the preceding Record Identification Codes.



RPG INPUT S

IBM International Business Machine Corporation

Program | Punching Instruction | Graphic | Punch

Programmer | Date

I

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1 Position	Not (N)	C/Z/D	Character	2 Position	Not (N)	C/Z/D	Character	Positio
01	I	CARDS	AA			Ø1	96		C	1					
02	I														
03	I														
04	I		BB			Ø2	96		C	2					
05	I														
06	I														
07	I														
08	I		CC			Ø3	96		C	3					
09	I														
10	I														
11	I		DD			Ø4									
12	I														

Figure 2-18. Unsequenced Record Types with Catch-All Sequence Entry

FIELD RECORD RELATION INDICATORS

You may have some programs which process several different record types. Two or more record types might contain identical fields. To eliminate coding these identical fields for every record type you may use the OR relationship which indicates that certain fields are found on all record types. Not all fields are identical in different record types, however. You must have some way of specifying those fields found on only specific record types in the OR relationship. Field record relation indicators indicate those fields found on only specific record types.

Field record relation indicators will relate:

- A field to a specific record type in the OR relationship.
- Control fields and split control fields to a specific record type in an OR relationship.
- Match fields for more than one record type (see *Match Fields and Multifile Processing*).

OR Relationship

You can eliminate duplicate coding by using an OR relationship to describe identical record types. This method also reduces the size of the program.

When using the OR relationship, you need to write the names of identical fields from more than one type of record only once on the Input Sheet. OR relationship specifications indicate that the fields named may be found on all of the record types. The following input specifications are necessary to set up the OR relationship:

1. Record identifying indicators (01-99) for each record type.
2. The letters OR in columns 14-15 for all record types other than the first.
3. Entries describing the record identification code of each record type (columns 21-41).

The record identifying codes must be described for *all* types of records in the file before any fields are described (Figure 2-19).

The letters OR are placed before the description of each record type except the first. OR indicates that the fields listed may be found on all record types. In this example, the fields listed may be found on records identified by an *N, D,* or *O* in column 96. Identical fields are described after the entries which establish the OR relationship.

OR Relationship With Field Record Relation Entries

In the example of printing a report by product class, all record types had identical fields (Figure 2-3). Suppose that the information on each record type is organized differently; the records have some fields which are identical and some which are not (Figure 2-20). Now you want to print only a description of new items. The record identified by an *N* is the only one with the DESC field. All record types still have CLASS, ITEMNO, DATE, and ONHAND fields.

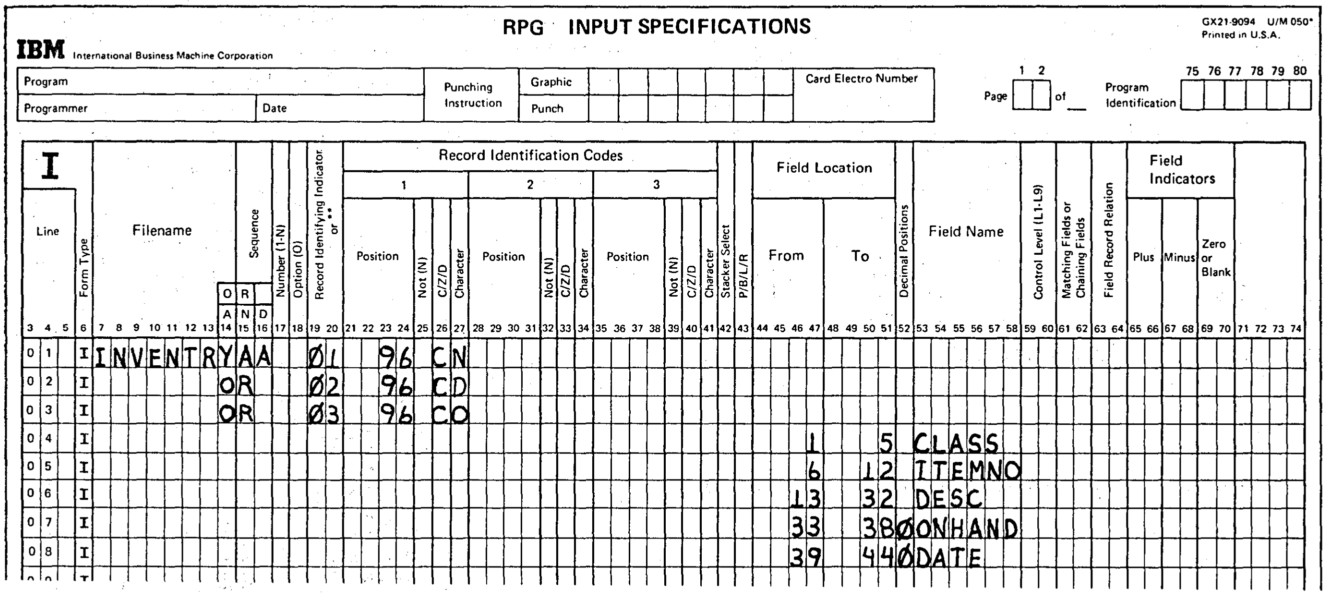
The OR relationship can be used when all fields are not identical. In this case, additional entries must be made in the field record relation columns (63-64) on the Input sheet. The entry consists of any of the record identifying indicators (01-99) assigned to a record type specified in the OR relationship. The record identifying indicator entered in columns 63-64 relates a field to a particular record by identifying the record type in which the field is found.

When columns 63-64 are blank, the fields listed are assumed to be found in the positions specified on *all* records in the OR relationship. When an entry is specified in columns 63-64, the field is found only on the record type having that record identifying indicator.

To use the OR relationship with field record relation entries you must:

1. Code the specifications describing record types in the OR relationship (Figure 2-21, lines 02, 03, and 04).
2. Describe all fields which are identical on all record types (Figure 2-21, lines 06, 07, and 08). In this example, the identical fields are CLASS, ITEMNO, and DATE.
3. Specify all fields that are found only on the first record type in the OR relationship, then the second record type, then the third, and so on (Figure 2-21, lines 10, 11, 12, and 13).

In this example, the only fields for the first record type which have not been described are DESC and ONHAND. For each field, the entry 01 must be made in columns 63-64. This entry means that DESC and ONHAND are found on only the record type 01 identified by an *N* in column 96.



IBM International Business Machine Corporation

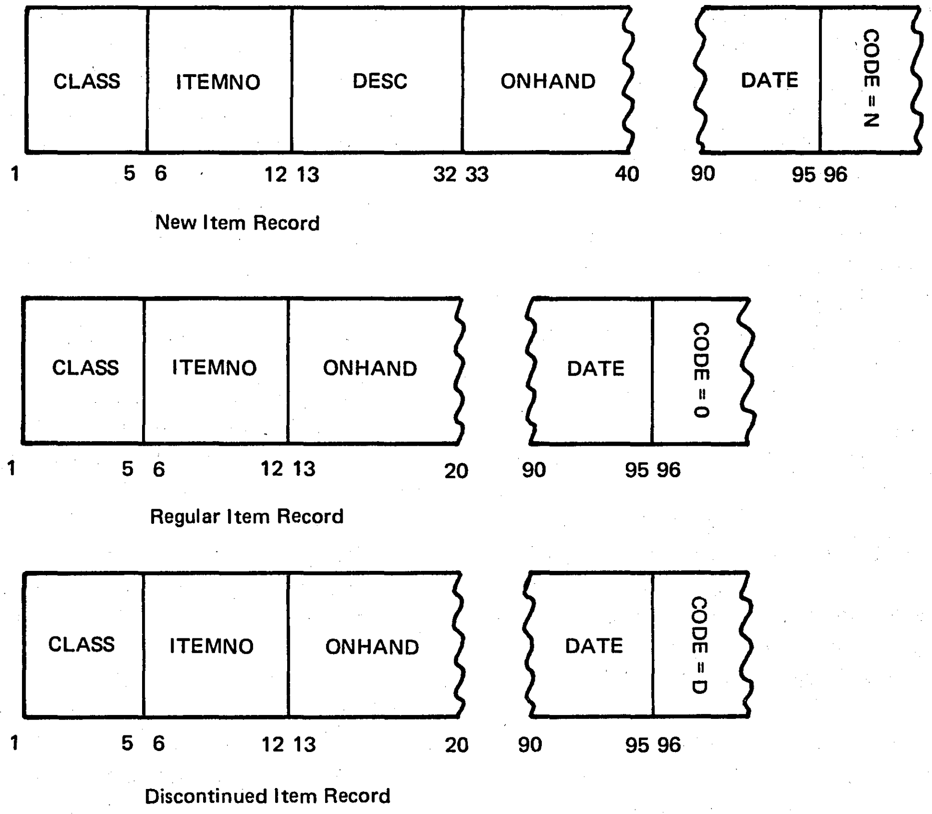
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page □□ of __
Program Identification □□□□□□

Line	Form Type	Filename	Sequence (OR/AND)	Record Identifying Indicator	Record Identification Codes 1: Position	C/Z/D	Character	Field Location: From	To	Decimal Positions	Field Name
01	I	INVENTRY	AA	Ø1	96	C	N				
02	I		OR	Ø2	96	C	D				
03	I		OR	Ø3	96	C	O				
04	I							1	5		CLASS
05	I							6	12		ITEMNO
06	I							13	32		DESC
07	I							33	38	Ø	ONHAND
08	I							39	44	Ø	DATE

Figure 2-19. Using the OR Relationship to Describe Identical Record Types

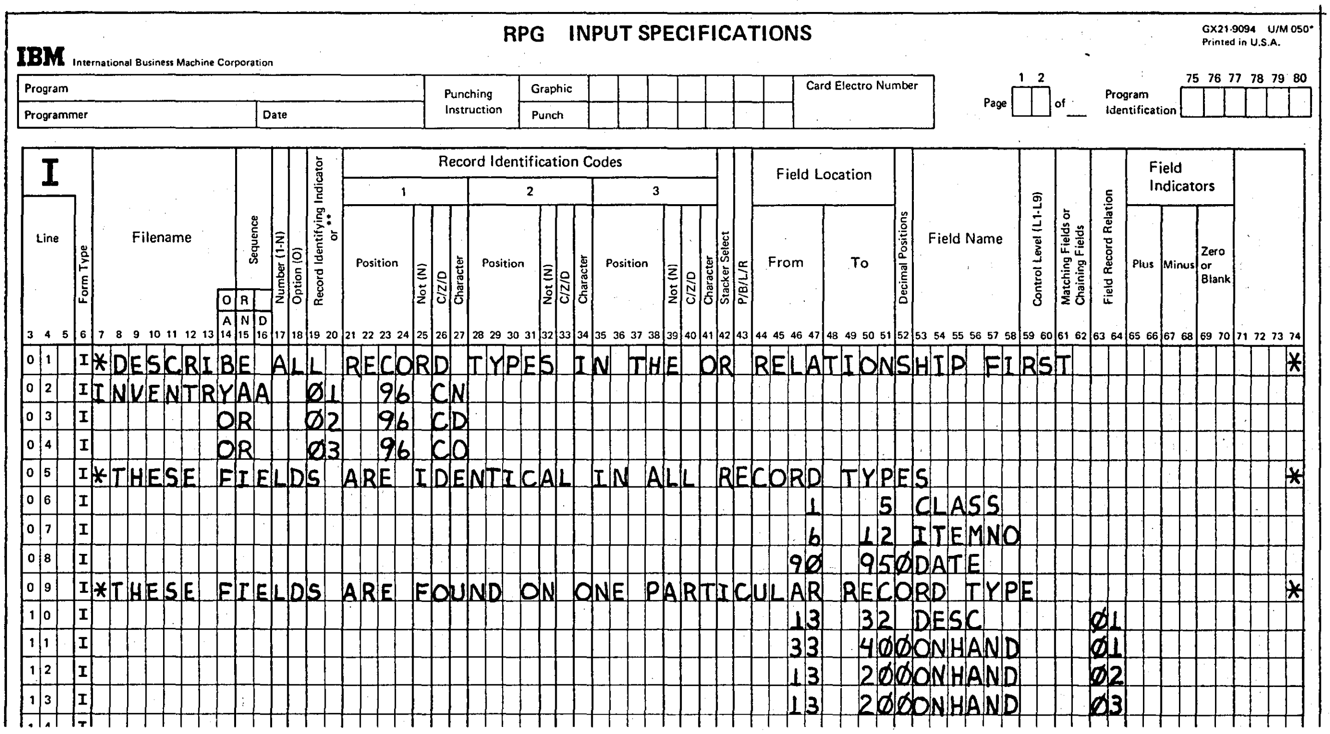


New Item Record

Regular Item Record

Discontinued Item Record

Figure 2-20. Record Types with Some Identical Fields



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

```
01 I*DESCRIBE ALL RECORD TYPES IN THE OR RELATIONSHIP FIRST            *
02 IINVENTRYAA  Ø1  96 CN
03 I       OR    Ø2  96 CD
04 I       OR    Ø3  96 CO
05 I*THESE FIELDS ARE IDENTICAL IN ALL RECORD TYPES                    *
06 I                                        1   5 CLASS
07 I                                        6  12 ITEMNO
08 I                                       9Ø  95ØDATE
09 I*THESE FIELDS ARE FOUND ON ONE PARTICULAR RECORD TYPE             *
10 I                                       13  32 DESC      Ø1
11 I                                       33  40ØONHAND    Ø1
12 I                                       13  20ØONHAND    Ø2
13 I                                       13  20ØONHAND    Ø3
14 I
```

Figure 2-21. Field Record Relation

The DESC field is related to the record identified by an *N* because this is the only record type having a DESC field. ONHAND, however, is found on all record types. ONHAND must be related to the record having an *N* in column 96 because it is in a different location on this record type. The field location of ONHAND must be specified and related to the corresponding record type by the record identifying indicators (Figure 2-21, line 11).

Remember that when fields are not identical on all record types, the field must be described and related to all record types on which it is found.

All fields relating to only one record type should be entered as a group and must be given the same record identifying indicators in column 63-64.

If most fields are common, describing the record type with field record relation usually reduces the number of specifications you must write and the amount of storage necessary to hold the instructions.

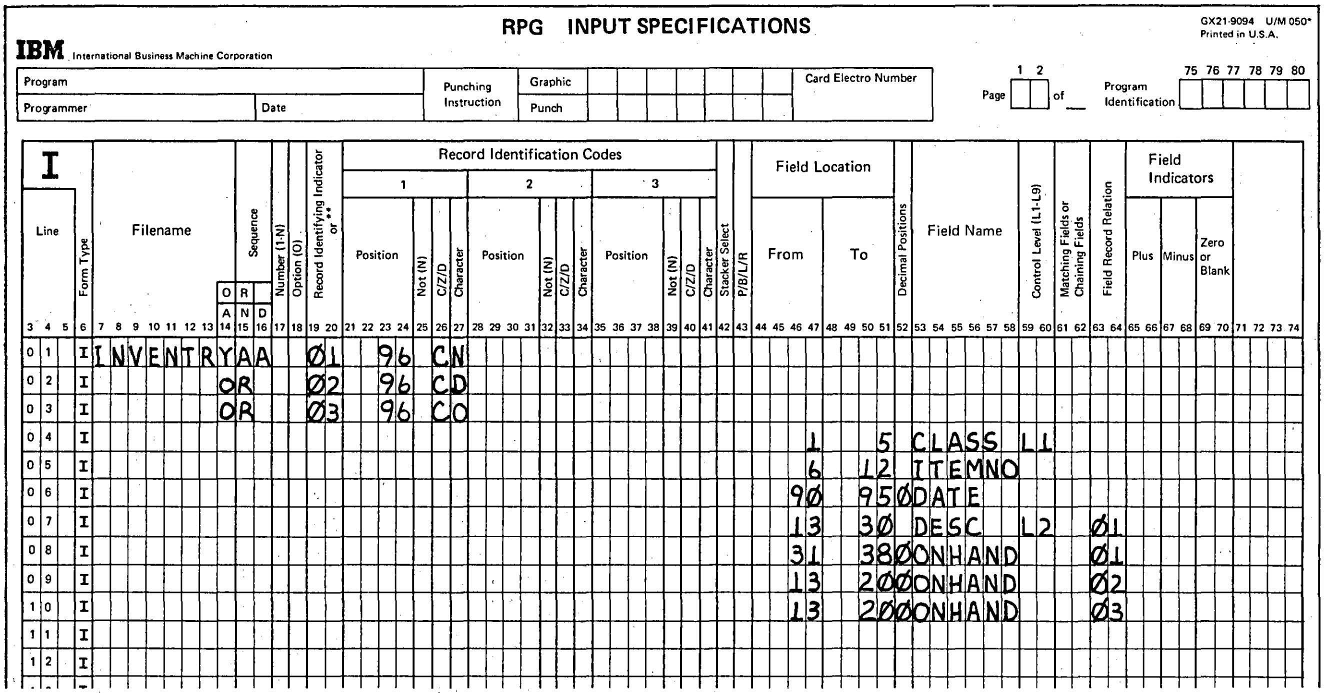
Field Record Relation with Control Fields

Control fields can also be related to a specific record type in an OR relationship by field record relation entries. In Figure 2-22 the CLASS field is a control field (L1 in columns 59-60). It is also found on all record types; blanks in the columns 63-64 indicate this. However, if a control field is found on only one record type, the control field must be related to the record type in which it is found by an entry in columns 63-64 (Figure 2-22, line 07).

The number of control fields need not be the same for every record in the OR relationship. Regardless of the number of control fields per record type, all control fields and all other fields related to the same record type should be entereu as a group (Figure 2-22, lines 07 and 08).

Field Record Relation with Split Control Fields

The rules applying to field record relation with control fields also apply to field record relation with split control fields. In addition, when split control fields are found on record types described in an OR relationship used with field record relation entries, all portions of the split control field must be assigned the same control level indicator and the same field record relation entry. This is necessary because all parts of a split control field are on the same record rather than on two different records.



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Programmer | Date

Line	Form Type	Filename	Sequence (OR/AND)	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank
01	I	INVENTRY	AA			Ø1	96		C	N																				
02	I		OR			Ø2	96		C	D																				
03	I		OR			Ø3	96		C	O																				
04	I																				1	5		CLASS	L1					
05	I																				6	12		ITEMNO						
06	I																				9Ø	95	Ø	DATE						
07	I																				13	3Ø		DESC	L2		Ø1			
08	I																				31	38	Ø	ONHAND			Ø1			
09	I																				13	2Ø	Ø	ONHAND			Ø2			
10	I																				13	2Ø	Ø	ONHAND			Ø3			
11	I																													
12	I																													

Figure 2-22. Field Record Relation with Control Fields

CONDITIONING USE OF INPUT FILES (EXTERNAL INDICATORS)

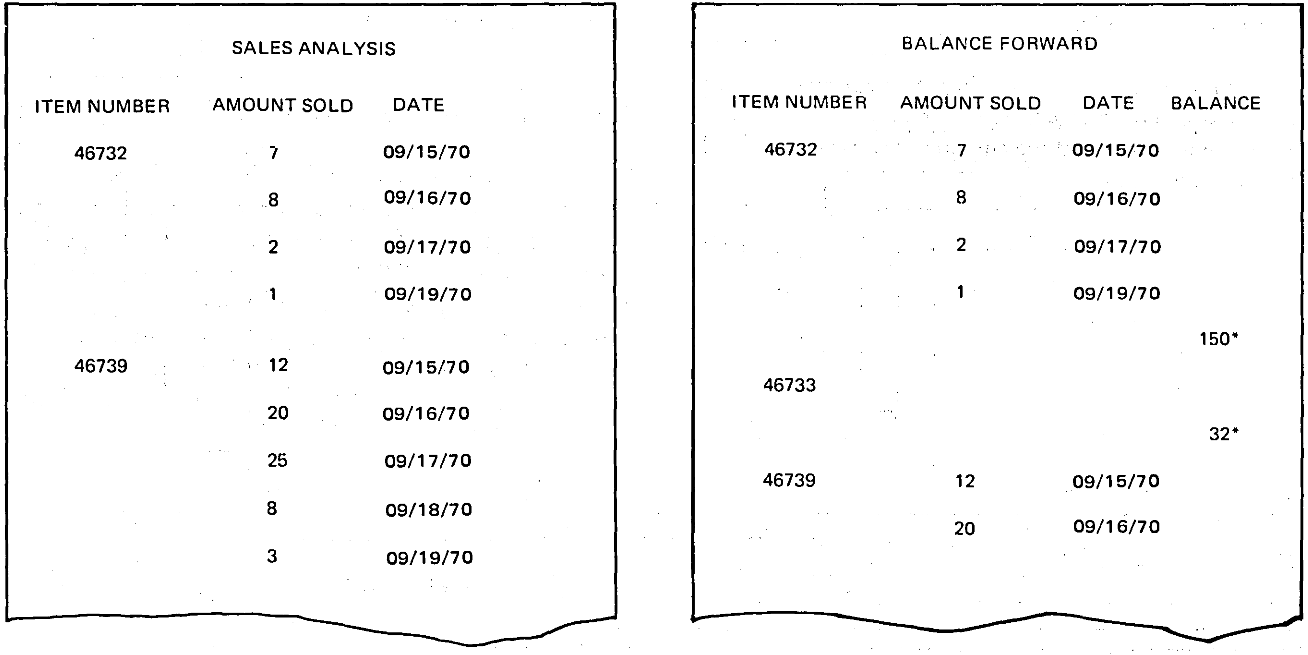
Thus far, in this chapter, you have read about jobs that require the complete processing of all input files specified. The following topics will illustrate how you can use RPG II to do jobs for which:

1. It is not necessary to process one or more of the files.
2. It is not necessary to process an *entire* file.

Using One Program to do More Than One Job

Have you ever thought how useful it would be if, when doing similar jobs, you could use one program to perform more than one function?

Consider, for example, the following jobs. Two types of reports are required each week. One is a sales analysis report showing what items sold during the week. The second is an inventory report showing balance on hand for each item in stock. Notice the similarity in the format of the reports (Figure 2-23).



```
                SALES ANALYSIS

ITEM NUMBER   AMOUNT SOLD    DATE

   46732          7        09/15/70

                  8        09/16/70

                  2        09/17/70

                  1        09/19/70

   46739         12        09/15/70

                 20        09/16/70

                 25        09/17/70

                  8        09/18/70

                  3        09/19/70
```

```
                  BALANCE FORWARD

ITEM NUMBER   AMOUNT SOLD     DATE     BALANCE

   46732          7         09/15/70

                  8         09/16/70

                  2         09/17/70

                  1         09/19/70

                                         150*
   46733

                                          32*
   46739         12         09/15/70

                 20         09/16/70
```

Figure 2-23. Two Similar Reports from Two Different Jobs

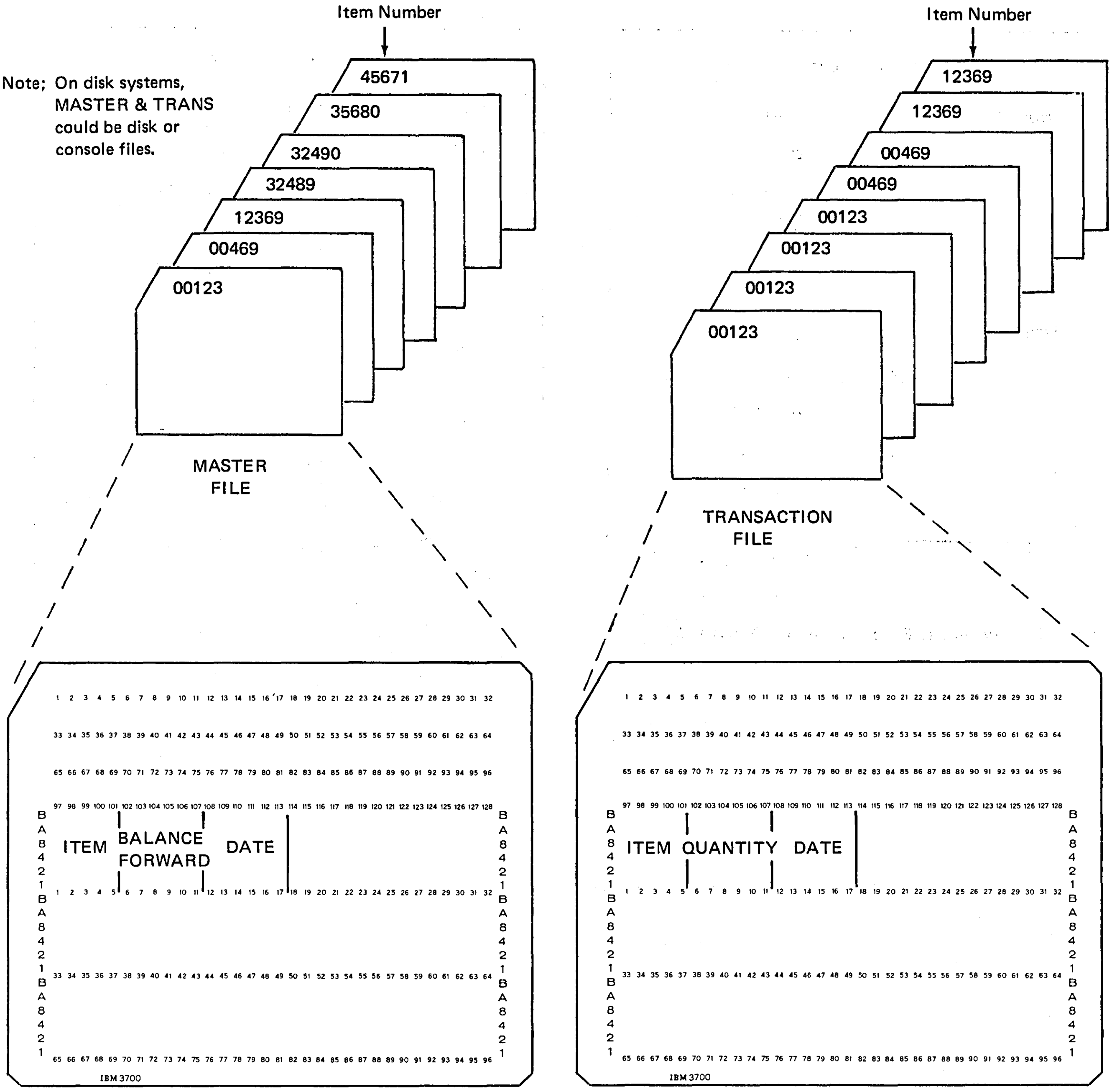
Two files are available: the MASTER file which contains balance forward records for all items in the store; and a transaction file (TRANS) which contains all the weekly sales for each item (Figure 2-24). Both files are in ascending order by item number.

The sales analysis report merely requires a listing of records found in the transaction file.

The inventory report requires that records from two files be matched. (If you are not familiar with multifile processing using two input files, see the chapter *Match Fields and Multifile Processing.*) When records from both files match, the number sold is subtracted from the balance on hand. The new balance is then printed on the report following the list of transactions.

One report requires two files; the other only one. How could you write one program to produce reports which have such different file requirements? If you had some way of telling the program when to expect the use of one file and when to expect two files, it could be done.

This you can do with external indicators.



Note; On disk systems, MASTER & TRANS could be disk or console files.

Figure 2-24. Format of Records Used to Produce Sales Analysis and Balance Forward Reports

Setting External Indicators

You are already familiar with several types of indicators used in the RPG II language. These indicators are used to:

1. Signal the occurrence of a specific condition, such as matching records, control break, or last record.

2. Control when certain operations should be performed; such as only when a control break occurs, or when a specific record type is read.

Most indicators are set by the program on the basis of the conditions which occur during the execution of the program. External indicators, however, are set by you prior to the execution of the program. You do this in one of the following ways, depending on which System/3 model you have:

Model 10 Card System: In order to set external indicators in the Card System, enter an Indicator Control Card in the System Initialization Program. The control card must have the following format:

Columns	*Entry*
1-2	// (two slashes)
3	blank
4-6	IND
7	blank
8-15	One-position entries indicating the setting of U1 through U8. Indicator U1 is set in column 8, U2 in column 9, and so on, as follows: 1 — Indicator is turned on 0 — Indicator is turned off ƀ (blank) — Indicator remains as it was set in the last job
16-96	blank

Figure 2-25 shows an Indicator Control Card which causes external indicators U1 and U8 to be set on, indicators U2 through U6 to be set off, and indicator U7 to remain as it was in the previous program.

Once an indicator is set, it is not changed during the entire program. The only way the setting may be changed for the next program is by another Indicator Control Card entered in the System Initialization Program.

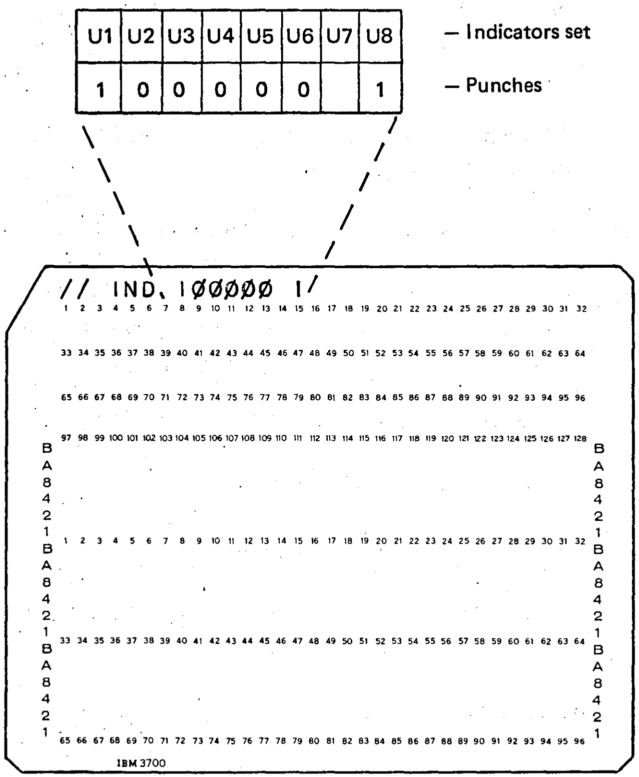


Figure 2-25. Indicator Control Card (Model 10 Card System)

Model 10 Disk System and Model 15: Although most indicators are set by the program, you set external indicators prior to the execution of the program. This is done by including a SWITCH statement in your Operational Control Language. The format of the SWITCH statement is:

```
// SWITCH indicator  settings
```

The indicator settings are:

1 = indicator is turned on.

0 = indicator is turned off.

X = indicator is unaffected.

Figure 2-26 shows a SWITCH statement which sets external indicators U1 and U8 on and indicators U2 through U6 off. Indicator U7 is unaffected.

Once an indicator is set, it is not changed until you provide another SWITCH statement or perform IPL. You cannot use the SETON or SETOF operation codes with external indicators.

On the Model 15, when operating in job mode, SWITCH settings are reset to 0 at end of job.

Model 6: The operator sets external indicators prior to execution of the program by responding to the SWITCH keyword displayed by the system. An eight-position response is possible, corresponding to the eight external indicators. Possible entries for each position are:

1 = indicator is turned on.

0 = indicator is turned off.

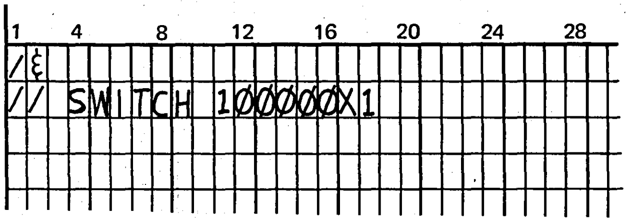
X = indicator remains unchanged.

For example, if the operator keys XXXX10XX in response to the SWITCH keyword:

- Indicator U5 is turned on.
- Indicator U6 is turned off.
- Indicators U1, U2, U3, U4, U7, and U8 remain unchanged.

While displaying the SWITCH keyword, the system displays the previous external indicator setting. If all indicators are to remain unchanged, the operator responds to SWITCH by pressing PROG START.

Indicators set by the SWITCH keyword retain their settings until another SWITCH statement changes them or the next IPL occurs.


1
4
8
12
16
20
24
28

```
/ξ
// SWITCH 1ØØØØØX1
```

Figure 2-26. SWITCH Control Statement

Using an External Indicator to Condition a File

You can assign an external indicator to a file. When the indicator is on, the file is used; when it is off, the file is not used. This then is how you can tell a program when to expect one file and when to expect two. Consider again the two jobs discussed previously: sales analysis and inventory.

The TRANS file is needed for both jobs, the MASTER file is only needed for the inventory job. Thus, the MASTER file is assigned the U1 indicator. You set the indicator on for the inventory job (MASTER is used here) and off for the sales analysis job (MASTER is not used here).

The U1 indicator is assigned to a file on the File Description sheet in columns 71-72. Any of the eight external indicators (U1-U8) could be used. U1 was arbitrarily chosen for this example (Figure 2-27).

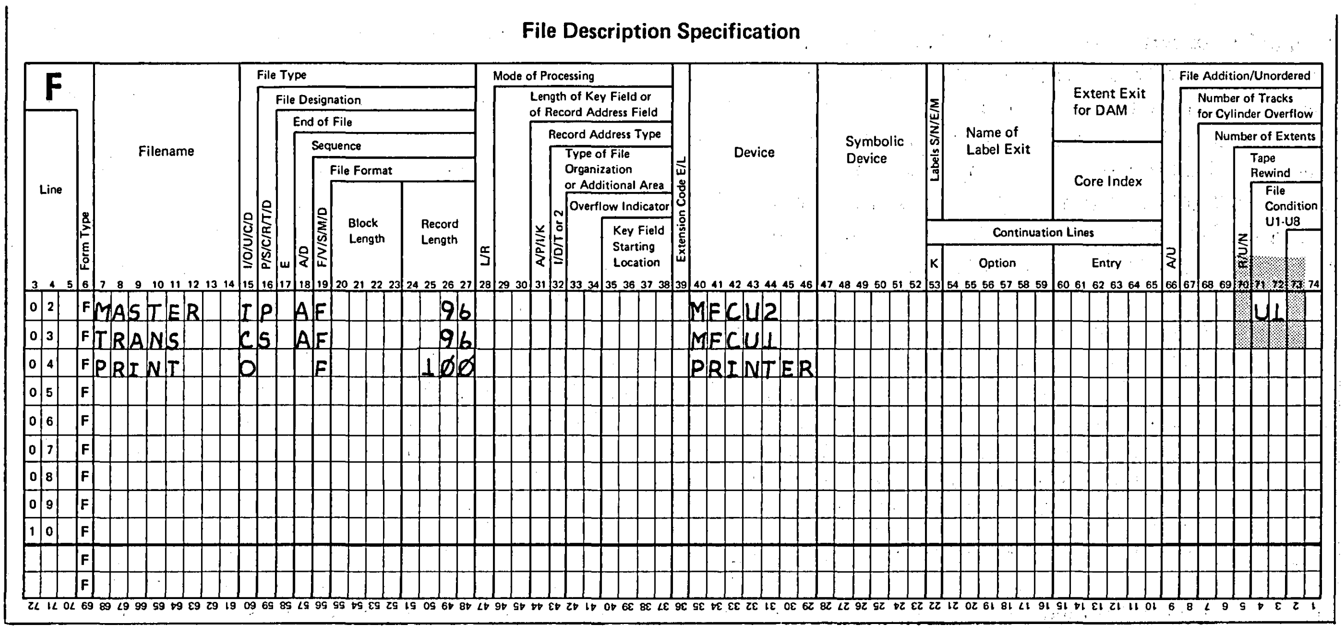
Naturally, the calculations performed and the type of report written out will depend upon which job is being done. Different calculation and output-format specifications are needed for each. In order to determine which specifications to use for a particular run of the program, calculation and output-format specifications must also be conditioned by the external indicator. This topic will be further discussed under *Controlling Operations in an RPG II Program.*

When writing a program which can do two jobs, be certain that the two jobs are very similar. Where the jobs require many different calculations and output operations, it would be easier to write two different programs than to use external indicators.

Ending the Program Before Processing All Files Completely

When should end-of-job operations take place? The program would normally end after all records have been processed. When you are reading one file, you usually want to process all records in that file. Normally, you wouldn't want to process a few records and then end the program unless, of course, you found an error condition. The computer also operates under the assumption that all records in the file should be processed before the program ends. The LR (last record) indicator which conditions end-of-job operations is not turned on until the last record has been processed.

Suppose, however, that you are using two files in your program. The computer assumes that all records in both files must be processed before the program ends. If you want the program to end before all records in both files are processed (for example, when the secondary file runs out of records), you can specify this. This is done by placing an E in column 17 of the File Description sheet for the file which will terminate the program.

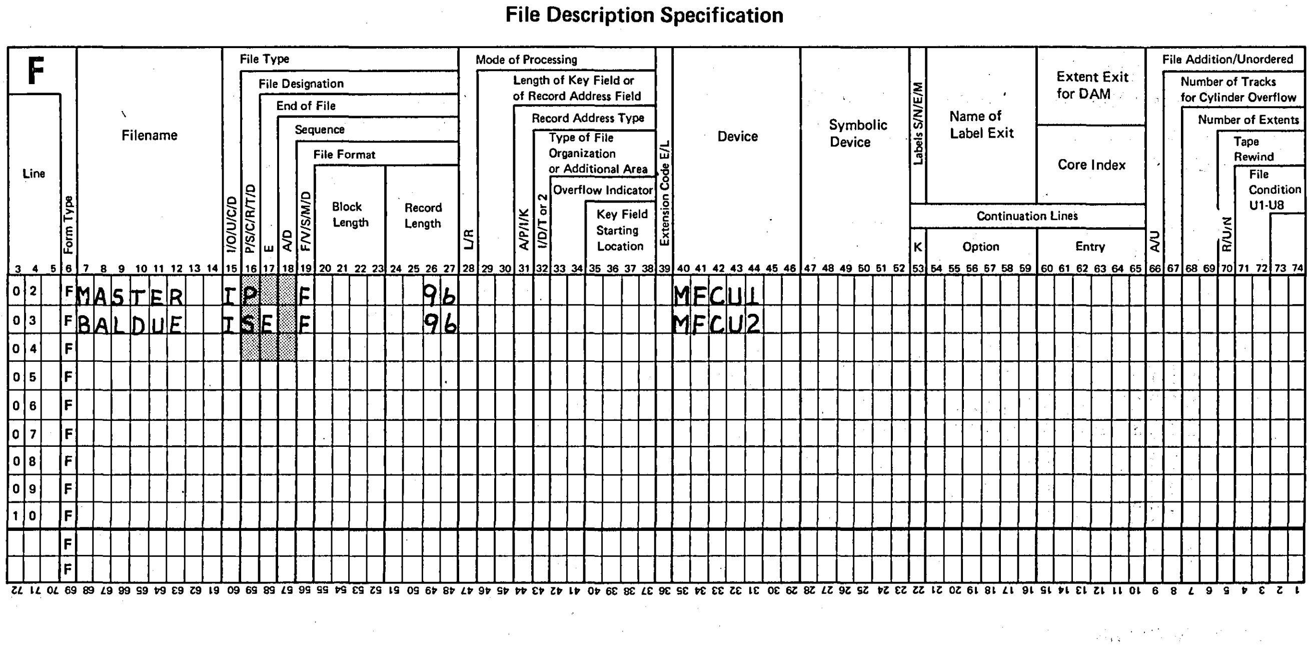


File Description Specification

Line	Form Type	Filename	File Type (I/O/U/C/D)	File Designation (P/S/C/R/T/D)	End of File (E)	Sequence (A/D)	File Format (F/V/S/M/D)	Record Length	Device	File Condition U1-U8
02	F	MASTER	I	P		A	F	96	MFCU2	U1
03	F	TRANS	C	S		A	F	96	MFCU1	
04	F	PRINT	O				F	100	PRINTER	
05	F									
06	F									
07	F									
08	F									
09	F									
10	F									

Figure 2-27. Assignment of an External Indicator

Figure 2-28 shows the File Description sheet used in a billing program that is to end when the last record of the secondary file has been processed. This is indicated by an E in column 17.

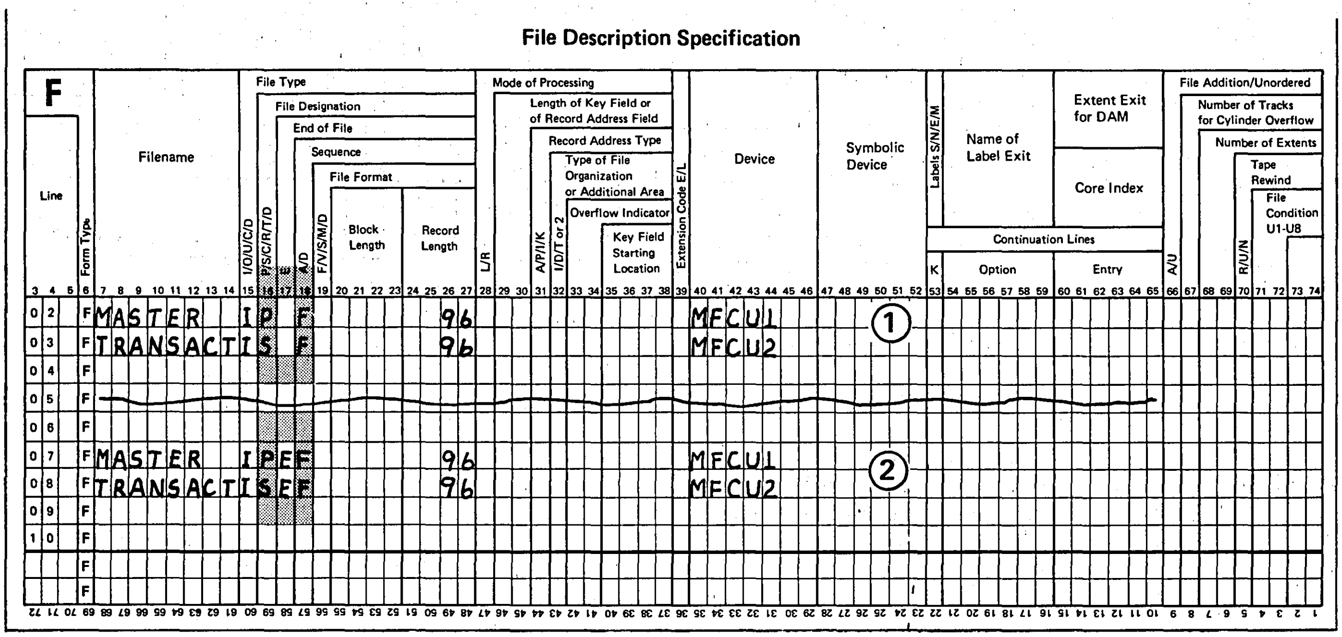


File Description Specification

Line	Form Type	Filename	File Type (I/O/U/C/D)	File Designation (P/S/C/R/T/D)	End of File (E)	Sequence (A/D)	File Format (F/V/S/M/D)	Record Length	Device
02	F	MASTER	I	P			F	96	MFCU1
03	F	BALDUE	I	S	E		F	96	MFCU2
04	F								
05	F								
06	F								
07	F								
08	F								
09	F								
10	F								

Figure 2-28. End-of-Job Specification

To indicate that the program will end only after all records from *all* files have been processed, you have the option of leaving column 17 blank for all input, update, or combined files or of placing an E in column 17 for all of these files. Figure 2-29 shows both ways of specifying that all files must be completely processed before end-of-job. For more information concerning end-of-job for programs using more than one file, see *Match Fields and Multi-file Processing.*



File Description Specification

Line	Form Type	Filename (7–14)	File Type (15)	File Designation (16)	End of File (17)	Sequence (18)	File Format (19)	Record Length (24–27)	Device (40–46)	
02	F	MASTER	I	P		F		96	MFCU1	①
03	F	TRANSACT	I	S		F		96	MFCU2	
04	F									
05	F									
06	F									
07	F	MASTER	I	P	E	F		96	MFCU1	②
08	F	TRANSACT	I	S	E	F		96	MFCU2	
09	F									
10	F									
	F									
	F									

Note: On disk systems, the files shown here might be on different devices, such as disk or console.

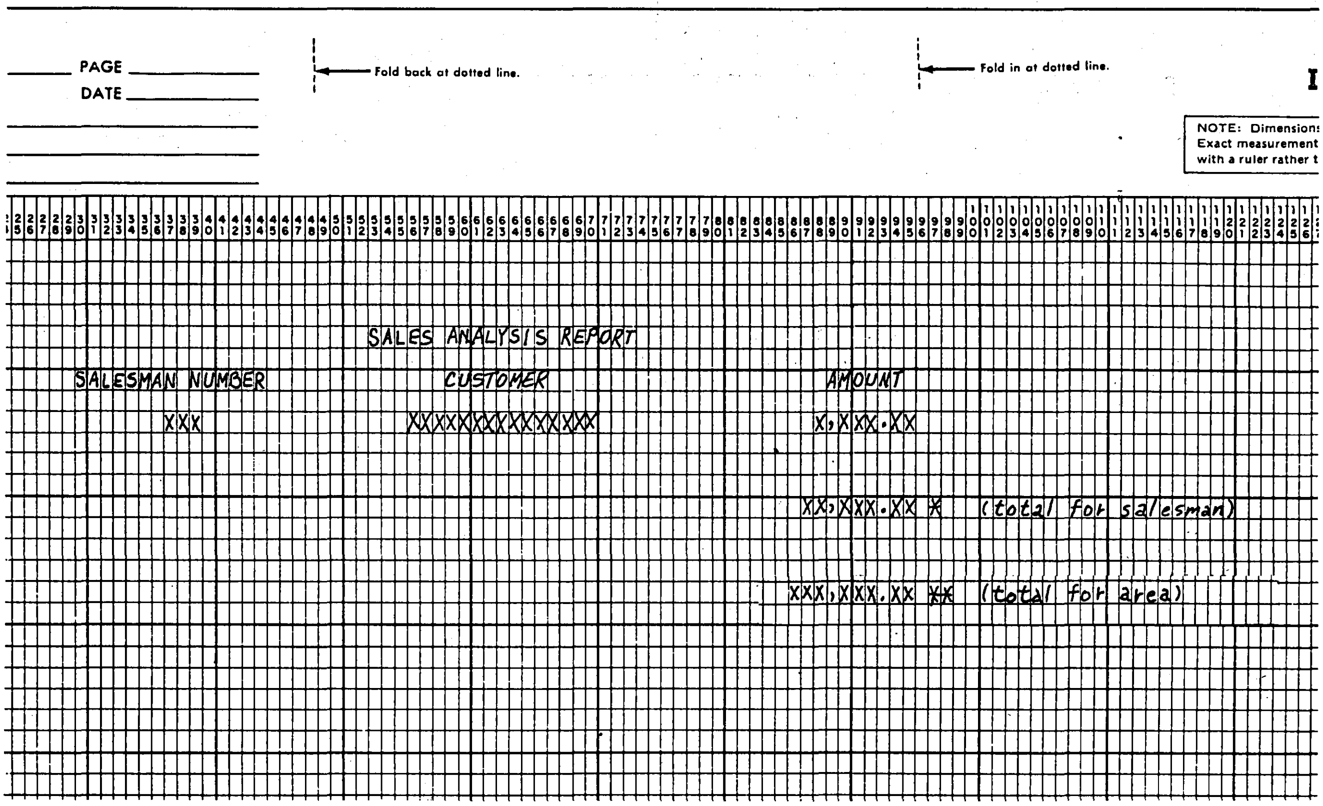
Figure 2-29. Two Ways of Specifying that All Records in All Files Must be Processed Before the Program Can End

1. A sales analysis report is to be group-indicated by salesman number as shown on the print chart below (GX20-1776).

   The fields on input file records are arranged as follows:

Positions	
1-2	Salesman number (last two digits of the three possible)
3-8	Amount of sale
9-23	Customer name
30	Salesman number (first digit of the three possible)
96	1 identifies the record type

   Fill in the input specifications for this program choosing your own file and field names.



______ PAGE ____________
DATE ____________

Fold back at dotted line.

Fold in at dotted line.

I

NOTE: Dimension
Exact measurement
with a ruler rather t

```
                               SALES ANALYSIS REPORT

       SALESMAN NUMBER               CUSTOMER                     AMOUNT

              XXX                XXXXXXXXXXXXXXX                  X,XXX.XX


                                                                XX,XXX.XX *   (total for salesman)


                                                               XXX,XXX.XX **  (total for area)
```

2. A large warehouse requires a weekly report showing the quantity of each item in stock. Three types of records are found in the file for every item in stock:
   a. *In Stock,* which records the quantity in stock at the beginning of the week. This record must be present, and there can be only one per item. It is identified by a Q in column 96.
   b. *Receipt,* which records the quantity brought into the warehouse. This record is optional. There may be several per item. It is identified by an I in column 96.
   c. *Issue,* which records the quantity shipped out of the warehouse. This record is optional. There may be several per item. It is identified by an O in column 96.

ITEMNO	DATE	INSTOK	
1 2 3 4 5 6 7 8	9 10 11 12 13 14	15 16 17 18 19 20 21	22 23 24 25 26 27 28 29 30 31

INSTOCK Record

ITEMNO	DATE	SHIPIN	
1 2 3 4 5 6 7 8	9 10 11 12 13 14	15 16 17 18 19 20 21	22 23 24 25 26 27 28 29 30 31

RECEIPT Record

ITEMNO	DATE	OUT	
1 2 3 4 5 6 7 8	9 10 11 12 13 14	15 16 17 18 19 20 21	22 23 24 25 26 27 28 29 30 31

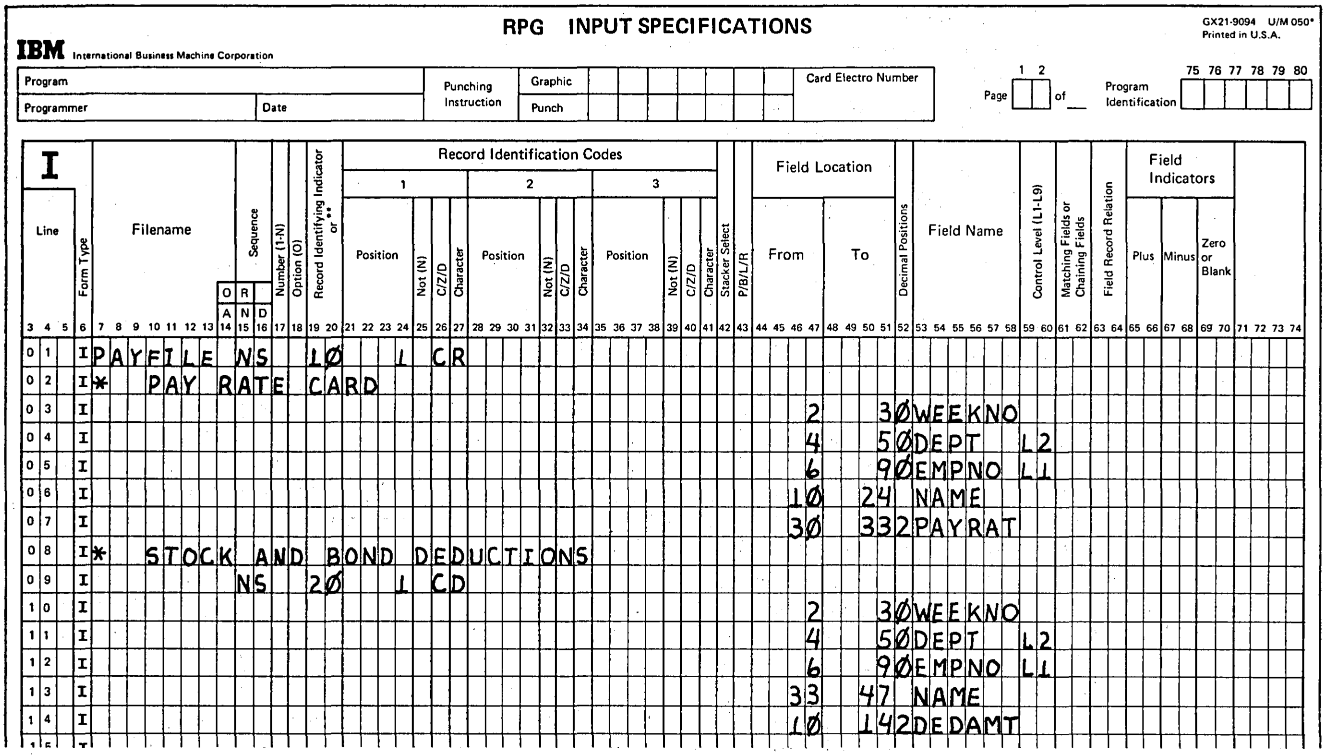
ISSUE Record

The records are grouped according to item number. All records of one item number must be in the order listed previously. Using the information given, write the Input specifications which are necessary to:

a. Check the sequence of the records in each group.

b. Prevent halting if unwanted or unused record types are read.

3. Rewrite the Input specifications shown below using field record relation entries.

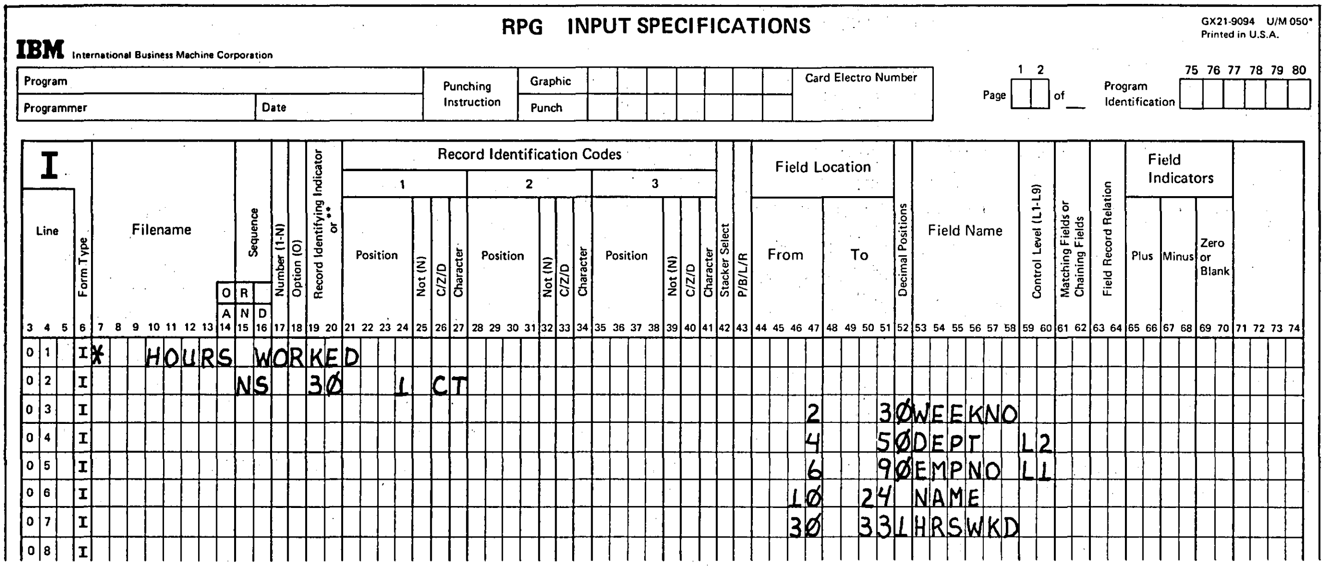


RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

```
01 IPAYFILE NS  10   1 CR
02 I*  PAY RATE CARD
03 I                                        2   30WEEKNO
04 I                                        4   50DEPT  L2
05 I                                        6   90EMPNO L1
06 I                                       10  24 NAME
07 I                                       30  332PAYRAT
08 I*  STOCK AND BOND DEDUCTIONS
09 I        NS  20   1 CD
10 I                                        2   30WEEKNO
11 I                                        4   50DEPT  L2
12 I                                        6   90EMPNO L1
13 I                                       33  47 NAME
14 I                                       10  142DEDAMT
15 I
```



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

```
01 I*  HOURS WORKED
02 I        NS  30   1 CT
03 I                                        2   30WEEKNO
04 I                                        4   50DEPT  L2
05 I                                        6   90EMPNO L1
06 I                                       10  24 NAME
07 I                                       30  331HRSWKD
08 I
```

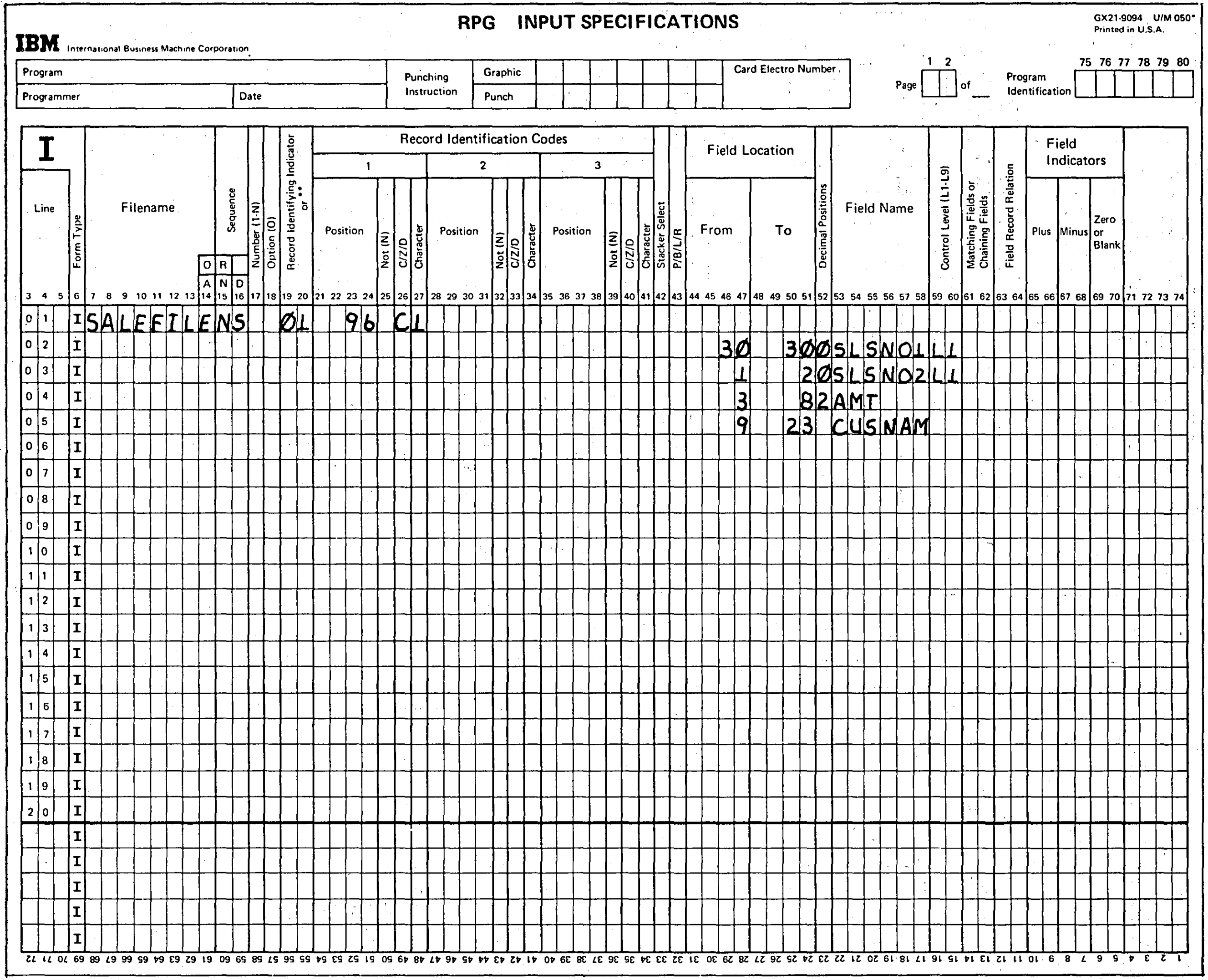
4. To write an RPG II program which will produce either of two similar reports requiring the use of one or two input files, you must enter a (an)____________ indicator in columns_______of the___________sheet for the optional file.

5. You are reading two files, NAMADD and TRANSACT. TRANSACT records will have a field added to them at output time. The program should stop when the last record from TRANSACT has been processed.

Make the necessary File Description entries for NAMADD and TRANSACT.

Answers To Review 2

1.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

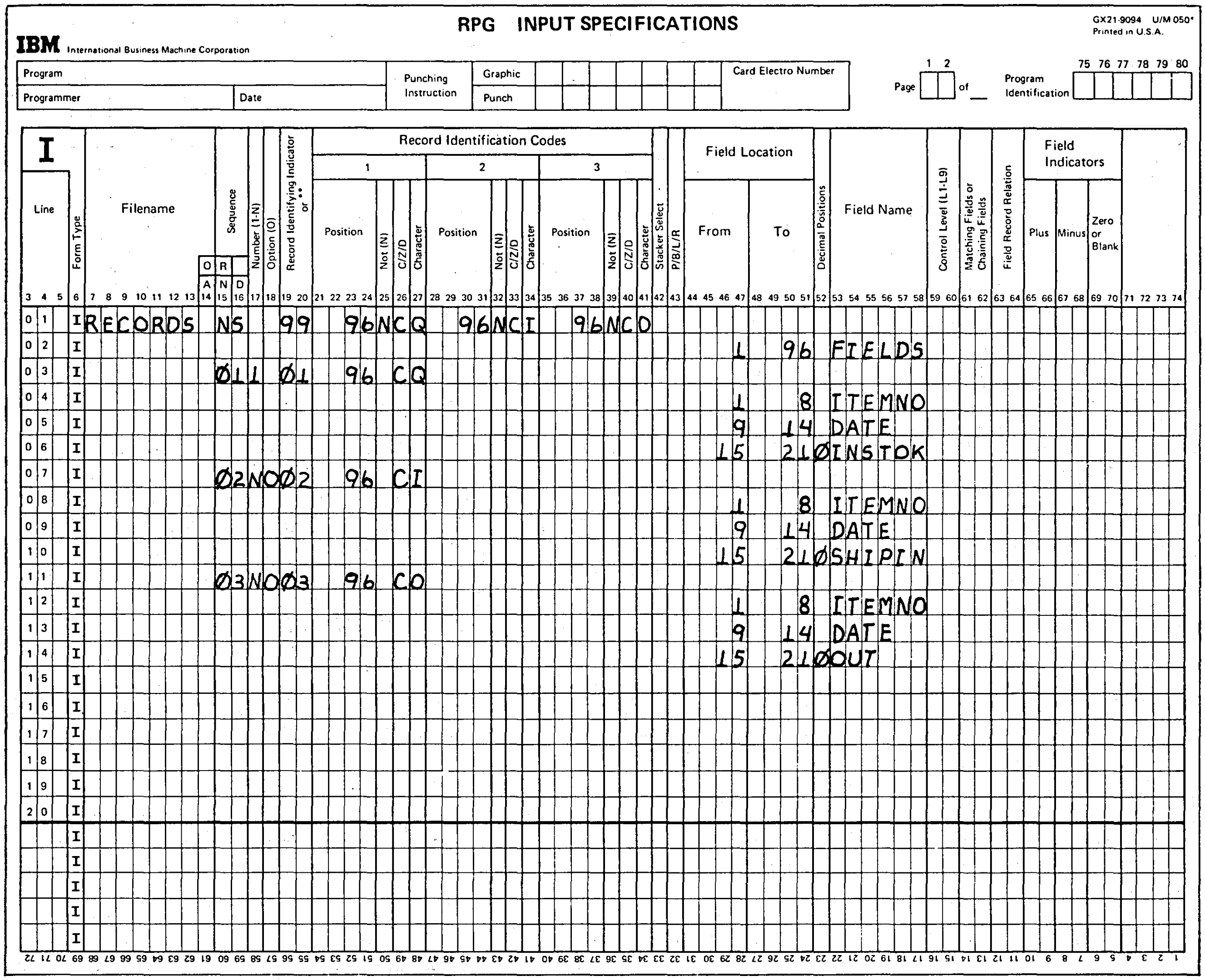
Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator (19-20)	Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)
01	I	SALEFILE	NS			Ø1	96		C	1										
02	I										3Ø	3Ø	Ø	SLSNO1	L1					
03	I										1	2	Ø	SLSNO2	L1					
04	I										3	8	2	AMT						
05	I										9	23		CUSNAM						
06	I																			
07	I																			
08	I																			
09	I																			
10	I																			
11	I																			
12	I																			
13	I																			
14	I																			
15	I																			
16	I																			
17	I																			
18	I																			
19	I																			
20	I																			
	I																			
	I																			
	I																			
	I																			
	I																			

The two fields which make up the salesman number should be assigned the same control level indicator to indicate that both fields are to be considered as one.

The split control fields must be specified on two adjacent lines. Since the first digit of the salesman number is in position 30, this single digit field should be specified before the field containing the last two digits of the number. The program determines the order in which the digits are to be arranged by the order in which the fields are specified.

2.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

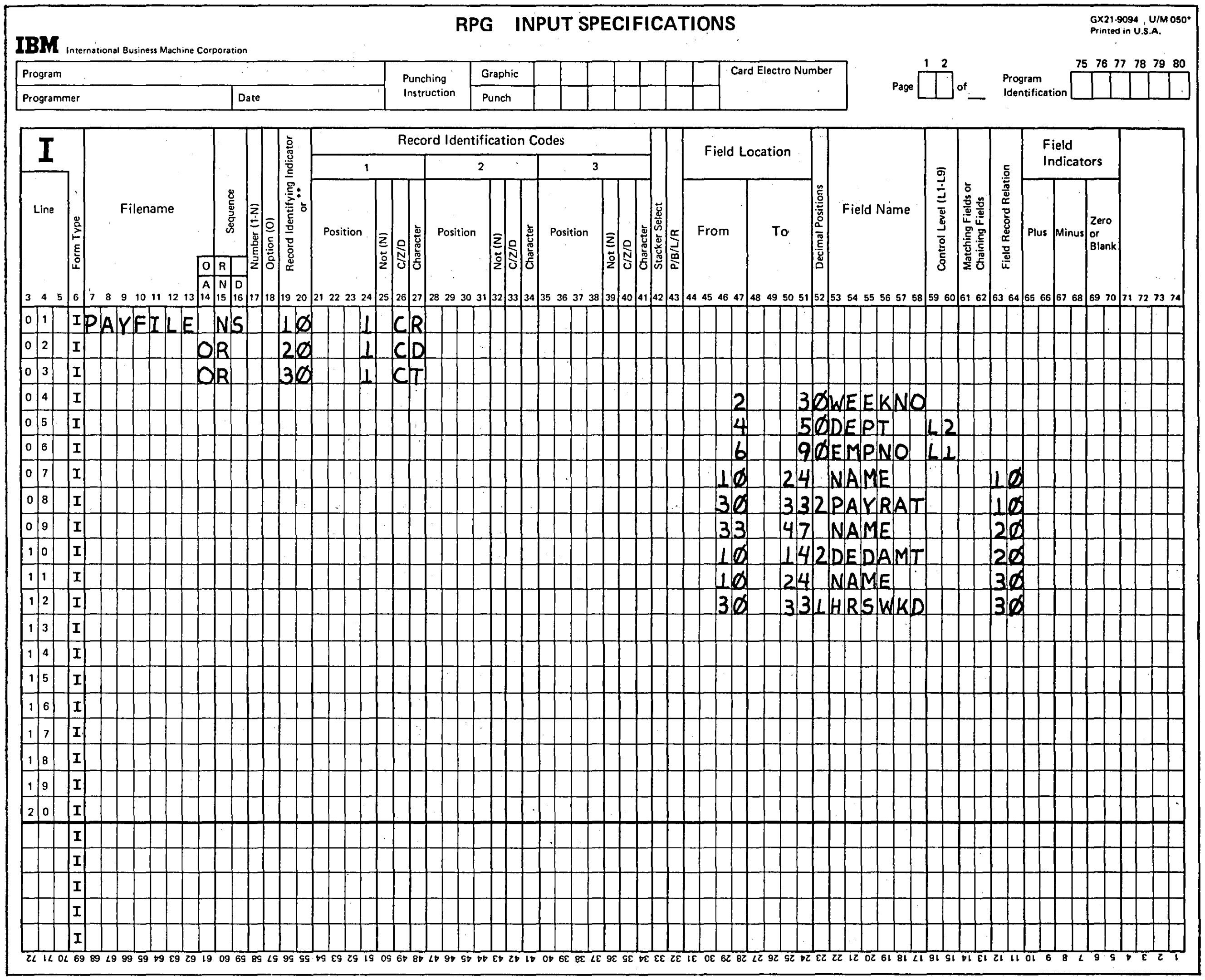
GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

```
01 IRECORDS NS  99  96NCQ  96NCI  96NCO
02 I                                        1  96 FIELDS
03 I        Ø11 Ø1  96 CQ
04 I                                        1   8 ITEMNO
05 I                                        9  14 DATE
06 I                                       15  21ØINSTOK
07 I        Ø2NOØ2  96 CI
08 I                                        1   8 ITEMNO
09 I                                        9  14 DATE
10 I                                       15  21ØSHIPIN
11 I        Ø3NOØ3  96 CO
12 I                                        1   8 ITEMNO
13 I                                        9  14 DATE
14 I                                       15  21ØOUT
15 I
16 I
17 I
18 I
19 I
20 I
```

Any alphabetic sequence entry must be entered first to catch all record types not being used for the program. This prevents halting when an unused or unidentified record is found in the input file. The three record types being sequence checked must be assigned sequence numbers in ascending order with the INSTOK record first, the RECEIPT record second, and the ISSUE record third. Since there is only one INSTOK record per group, a 1 must be entered in column 17. This record must be present so column 18 is left blank. Both of the remaining record types require an N in column 17 and an O in column 18. They are optional, and there may be more than one per group. Any record identifying indicators (01-99) you choose are correct.

3.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program

Programmer | Date

Punching Instruction | Graphic | Punch

Card Electro Number

Page 1 2 of

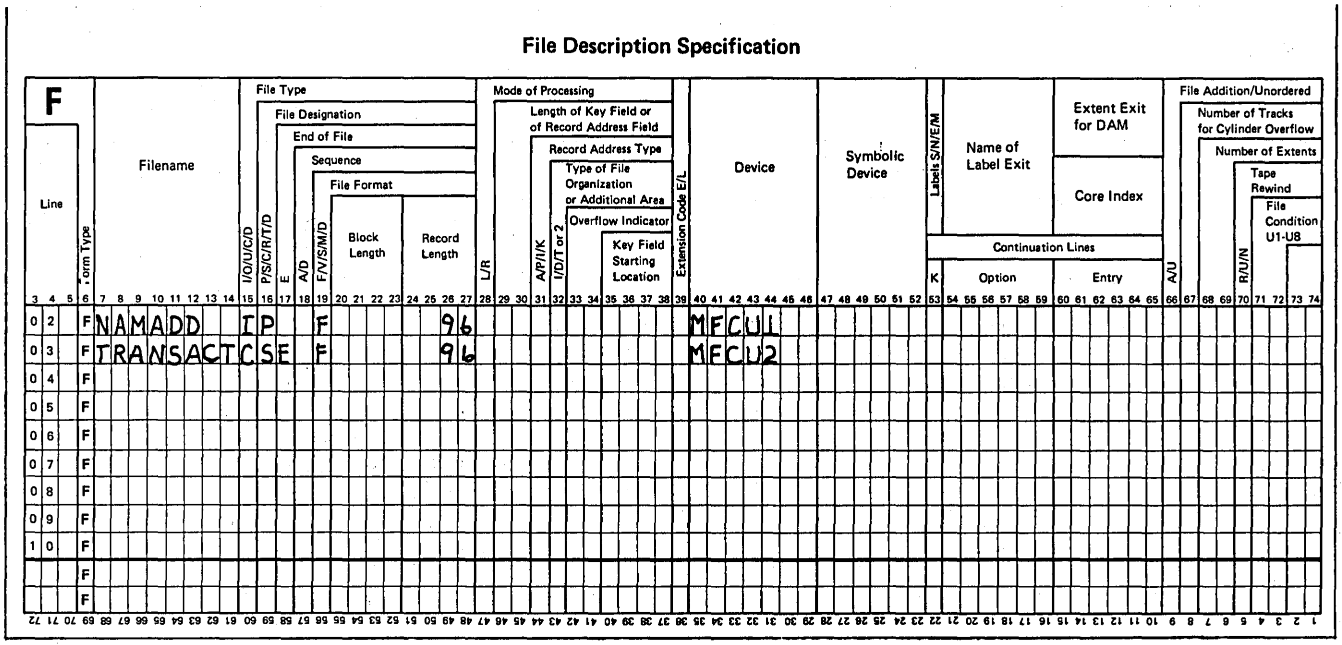
Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence (OR/AND)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Plus	Minus	Zero or Blank
01	I	PAYFILE	NS	1Ø	1		C	R										
02	I		OR	2Ø	1		C	D										
03	I		OR	3Ø	1		C	T										
04	I								2	3	Ø	WEEKNO						
05	I								4	5	Ø	DEPT	L2					
06	I								6	9	Ø	EMPNO	L1					
07	I								1Ø	24		NAME			1Ø			
08	I								3Ø	33	2	PAYRAT			1Ø			
09	I								33	47		NAME			2Ø			
10	I								1Ø	14	2	DEDAMT			2Ø			
11	I								1Ø	24		NAME			3Ø			
12	I								3Ø	33	1	HRSWKD			3Ø			
13	I																	
14	I																	
15	I																	
16	I																	
17	I																	
18	I																	
19	I																	
20	I																	

Because these record types contain common fields, the OR relationship may be used to describe them. However, since not all fields are common to all record types, field record relation entries must also be used. All common fields—WEEKNO, EMPNO, and DEPT—are described first. The NAME field, although found on all record types, is in different locations. Thus, it must be related to all record types by specifying it and its end position three times and using the record identifying indicator in columns 63-64 to indicate the record type with which it is associated. PAYRAT is found in only record type 10. Thus 10 is placed in the Field Record Relation columns (63-64). DEDAMT and HRSWKD are related to the record type on which they are found in the same way. Remember that all fields related to one record type must be grouped together.

4. An external indicator (U1-U8); 71-72; File Description.

5.



File Description Specification

```
  02 FNAMADD  IP  F      96            MFCU1
  03 FTRANSACTCSE F      96            MFCU2
  04 F
  05 F
  06 F
  07 F
  08 F
  09 F
  10 F
     F
     F
```

Line
Form Type
Filename
File Type
File Designation
End of File
Sequence
File Format
I/O/U/C/D
P/S/C/R/T/D
E
A/D
F/V/S/M/D
Block Length
Record Length
Mode of Processing
Length of Key Field or of Record Address Field
Record Address Type
Type of File Organization or Additional Area
Overflow Indicator
Key Field Starting Location
L/R
A/P/I/K
I/D/T or 2
Extension Code E/L
Device
Symbolic Device
Labels S/N/E/M
Name of Label Exit
Extent Exit for DAM
Core Index
Continuation Lines
K
Option
Entry
A/U
File Addition/Unordered
Number of Tracks for Cylinder Overflow
Number of Extents
Tape Rewind
R/U/N
File Condition U1-U8

In the example above, TRANSACT is specified as a combined MFCU file because it is to be both read and punched. An E is entered in column 17 for this same file to indicate that the program should end when all TRANSACT records have been processed.

If you have a disk system, however, TRANSACT could be a disk update file (U in column 15). NAMADD would probably also be a disk file. An E would still be entered in column 17 for the TRANSACT file.

Chapter 3. Controlling Printer Output

CHAPTER 3 DESCRIBES:

RPG II overflow and fetch overflow to control page formatting.

RPG II fetch overflow object program cycle.

Aligning printer forms.

Editing with edit words.

Using the special RPG II word, *PLACE, to print duplicate information.

Dual printer files.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

The RPG II object program cycle for overflow.

Function of RPG II indicators.

Using edit codes to punctuate numeric data.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

RPG II overflow and fetch overflow.

Effects of fetch overflow on the RPG II object program cycle.

Aligning printer forms.

Using edit words.

*PLACE

Coding for the Dual Feed Carriage Feature on the IBM 5203 Printer and coding for Dual Feed Tractors on the IBM 2222 Printer.

Note: You can use the review questions contained in *Review 3* at the end of this chapter to test your comprehension of each topic in the chapter. Questions are grouped according to the topic to which they apply. Answers follow the review questions.

INTRODUCTION

The most important part of any RPG II program is the result—the output. This chapter describes the RPG II coding necessary to format and punctuate printed output to make it easier to read and understand. Methods are also described for duplicating information on the output record and using dual printer files to print two reports in the same program.

Using the printer, you can create a report consisting of individual lines (records) recorded consecutively on stock paper. You may also use it to record information on your own preprinted forms, such as on bills, invoices, and checks. Regardless of the paper or form you are printing on, you are always interested in obtaining a report that is neat and readable. This means that the format of the report and data on the report must be considered when planning the program.

RPG II coding for the different printers available on System/3 is nearly identical. Differences in coding are noted. The available printers are:

- IBM 5203 Printer (Model 10)
- IBM 1403 Printer (Model 10 Disk System and Model 15)
- IBM 5213 Printer (Model 6)
- IBM 2222 Printer (Model 6)

USING OVERFLOW AND FETCH OVERFLOW TO CONTROL PAGE FORMATTING

RPG II performs automatic page formatting. With standard 66 line forms, it leaves five blank lines at the top of a page and six at the bottom. (Six lines are printed per inch; eight lines per inch are also possible.) However, automatic page formatting may not always meet your needs. If you want control over page formatting, you can use an overflow indicator (OA-OG, OV). For instance, assume that at the end of every month you prepare an inventory report which consists of a list of the quantity of all items in stock by product class. Items are listed by product class, and each product class should start on a new page (Figure 3-1).

Suppose the heading were to start on line 11 of each page. To have an equal margin (ten spaces) on top and bottom, line 56 should be the last printed line on the page (assuming 66 lines per page). For this report, you must use an overflow indicator to control page format.

Overflow Indicators

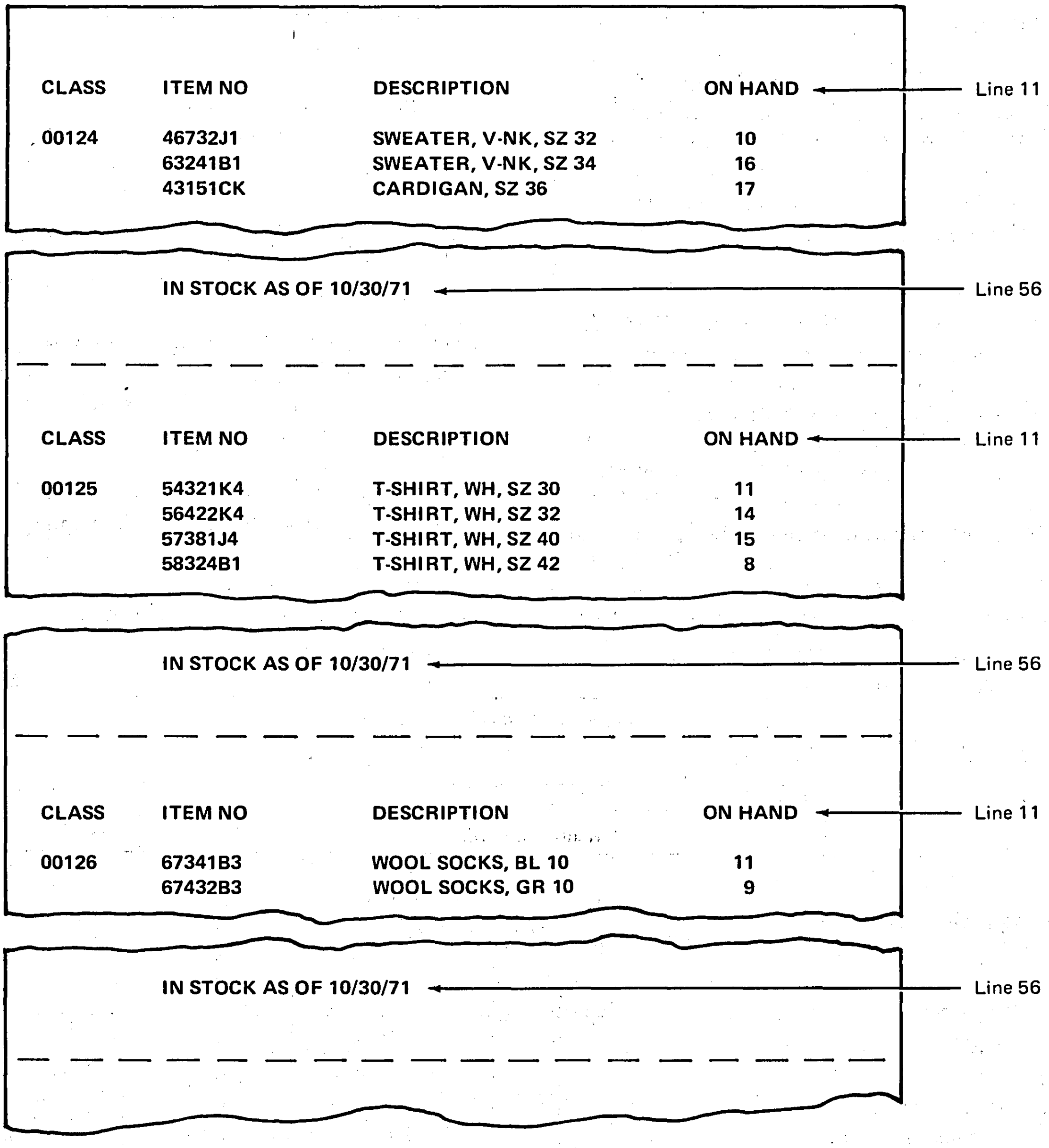
Overflow indicators, like other indicators, are used to do two things:

- Signal a certain condition.
- Control when specific operations (including those which control page format) are performed.

For example, in the monthly inventory report, items in stock are listed by product class. The report consists of 46 lines per page (starting line is 11 and ending line 56). Some product classes are going to have more than 46 different items in stock. For these classes, additional pages (overflow pages) are required to list the items.

Normally, the *overflow line* is the last line you want to print on the page. For this report, the overflow line would be line 56. When this line is printed, the overflow indicator (if one is assigned) is turned on to signal that the last line you wished printed on the page has been reached.

When the overflow indicator is on, you know that the overflow line has been reached. At the end of the page, operations, such as advancing to a new page (the overflow page) and printing headings on the new page, can be performed. By assigning and using overflow indicators, you can print special lines at the bottom of the page and at the top of the new page. Because you do these operations only when the overflow indicator is on, you will have to condition these operations by the overflow indicator.



```
CLASS     ITEM NO          DESCRIPTION                  ON HAND

00124     46732J1          SWEATER, V-NK, SZ 32            10
          63241B1          SWEATER, V-NK, SZ 34            16
          43151CK          CARDIGAN, SZ 36                 17

          IN STOCK AS OF 10/30/71

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CLASS     ITEM NO          DESCRIPTION                  ON HAND

00125     54321K4          T-SHIRT, WH, SZ 30              11
          56422K4          T-SHIRT, WH, SZ 32              14
          57381J4          T-SHIRT, WH, SZ 40              15
          58324B1          T-SHIRT, WH, SZ 42               8

          IN STOCK AS OF 10/30/71

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CLASS     ITEM NO          DESCRIPTION                  ON HAND

00126     67341B3          WOOL SOCKS, BL 10               11
          67432B3          WOOL SOCKS, GR 10                9

          IN STOCK AS OF 10/30/71

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 3-1. End-of-Month Inventory Report

Specifications for Using Overflow Indicators

You must specify to the RPG II compiler how reports should be printed. To tell it what to do, you make line counter, file description, and output-format specifications.

Line Counter Specifications

Line counter specifications, found on the bottom half of the Extension and Line Counter sheet (Figure 3-2), are used exclusively for defining the number of lines you want printed on each page.

Every time you use an overflow indicator to control formatting, you should prepare line counter specifications. Otherwise, a page length of 66 lines will be assumed with line 60 as overflow line.

Figure 3-3 is a sample Line Counter sheet for the inventory report. Columns 7-14 are for filename. Only a *printer* file name can be used here. Columns 15-22 contain the entries for report formatting:

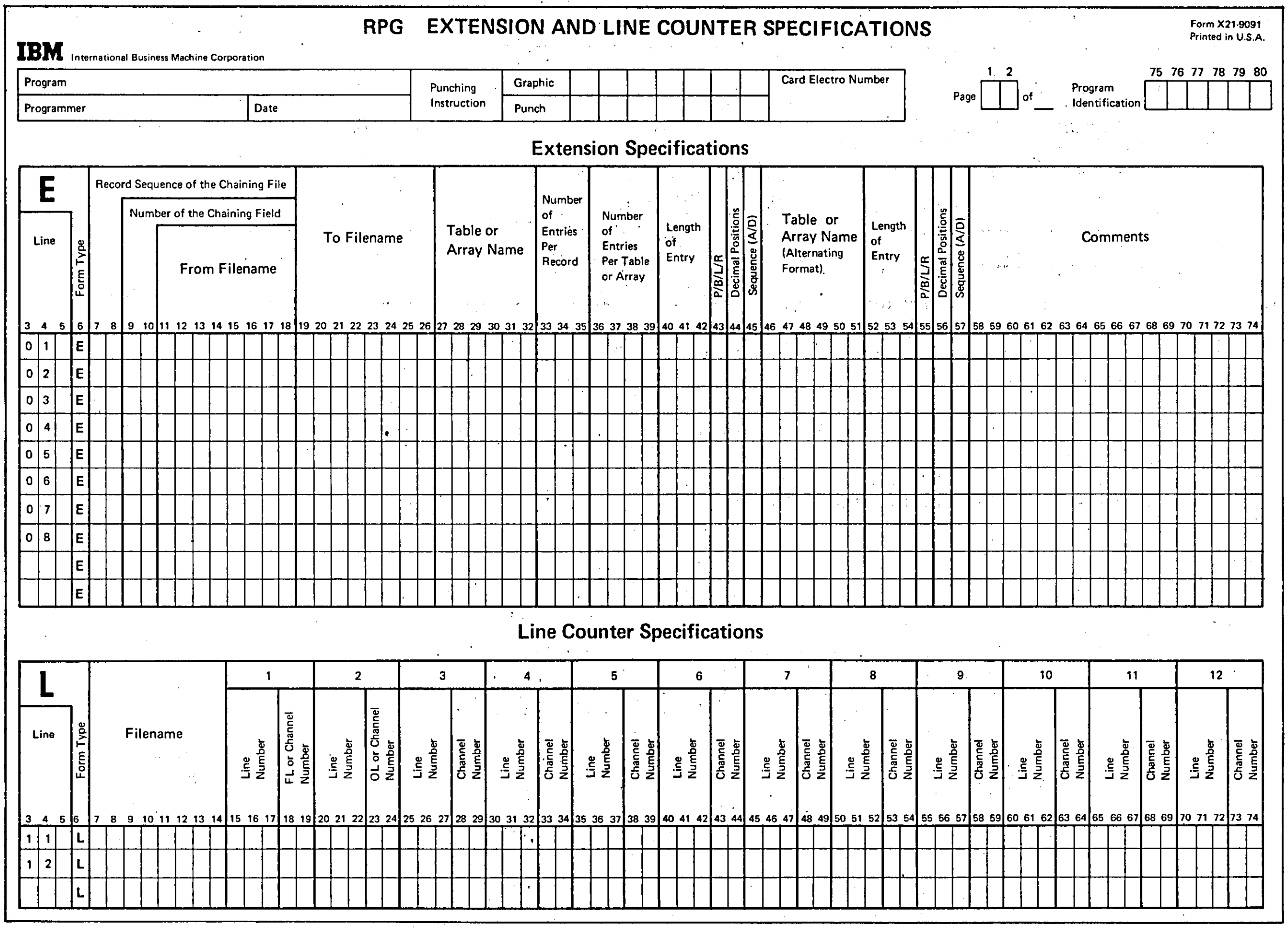
- Columns 15-17: Place in these columns the number of available lines per page. Your page can contain a maximum of 112 lines. The inventory report uses standard 11 inch paper, providing 66 lines per page.

- Columns 18-19: Put the letters FL in these columns to show that the previous specifications gave form length.

- Columns 20-22: Enter in these columns the number of the overflow line, when you want the overflow indicator to be turned on. In the example given, it was 56. You can use any number from 1-112.

- Columns 23-24: Enter the letters OL in these columns to show that the previous specification was the overflow line.

Notice that columns 25-80 are not used.



RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

IBM International Business Machine Corporation

Form X21-9091
Printed in U.S.A.

Program		Punching Instruction	Graphic		Card Electro Number
Programmer	Date		Punch		

Page 1 2 □□ of ___ Program Identification 75 76 77 78 79 80 □□□□□□

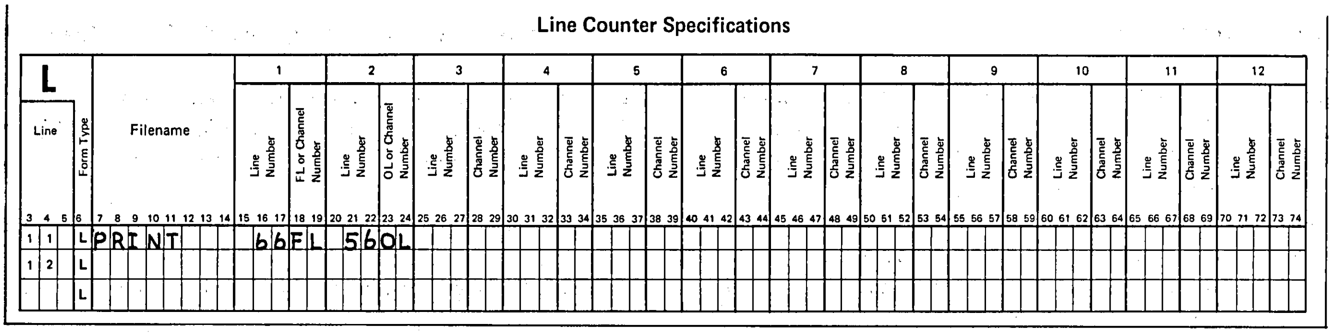
Extension Specifications

Line	Form Type		Record Sequence of the Chaining File / Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E																	
0 2	E																	
0 3	E																	
0 4	E																	
0 5	E																	
0 6	E																	
0 7	E																	
0 8	E																	
	E																	
	E																	

Line Counter Specifications

Line	Form Type	Filename	1		2		3		4		5		6		7		8		9		10		11		12	
			Line Number	FL or Channel Number	Line Number	OL or Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
3 4 5	6	7 8 9 10 11 12 13 14	15 16 17	18 19	20 21 22	23 24	25 26 27	28 29	30 31 32	33 34	35 36 37	38 39	40 41 42	43 44	45 46 47	48 49	50 51 52	53 54	55 56 57	58 59	60 61 62	63 64	65 66 67	68 69	70 71 72	73 74
1 1	L																									
1 2	L																									
	L																									

Figure 3-2. RPG II Extension and Line Counter Specification Sheet



Line Counter Specifications

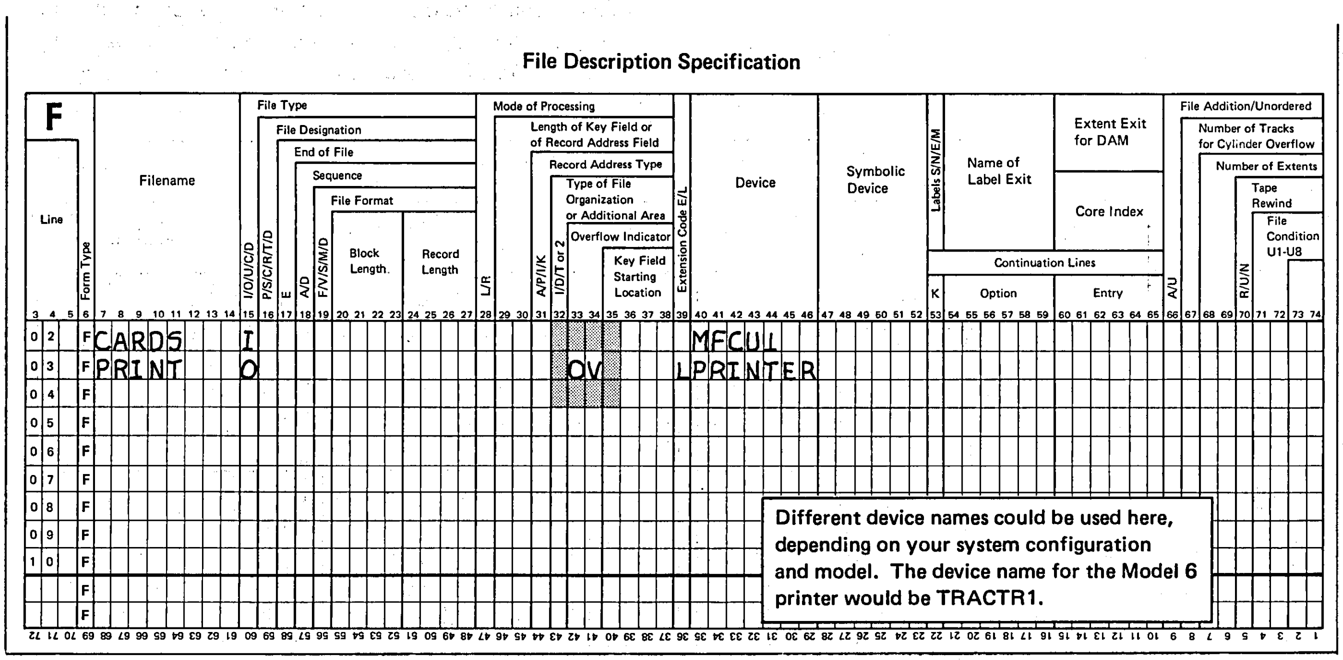
L			1		2		3		4		5		6		7		8		9		10		11		12	
Line	Form Type	Filename	Line Number	FL or Channel Number	Line Number	OL or Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
11	L	PRINT	66	FL	56	OL																				
12	L																									
	L																									

Figure 3-3. Line Counter Specifications

File Description Specifications

You must assign an overflow indicator to the *printer file* when you want to control the format of printed reports. This is done by an entry in columns 33-34 of the File Description sheet (Figure 3-4). You may choose to enter any of the following overflow indicators: OA, OB, OC, OD, OE, OF, OG, or OV. The one you choose, however, must be used throughout the program. *L* must also be entered in column 39 to indicate that line counter specifications are used.

These two entries indicate to the RPG II compiler that it should not provide automatic page formatting, but should format according to your specifications. If you do not make an entry in columns 33-34 of the File Description sheet for a printer file, an automatic skip to line 1 occurs on overflow.



File Description Specification

F			File Type	File Designation	End of File	Sequence	File Format			Mode of Processing																		
Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V/S/M/D	Block Length	Record Length	L/R	Length of Key Field or of Record Address Field	Record Address Type A/P/I/K	Type of File Organization or Additional Area I/D/T or 2	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels S/N/E/M K	Name of Label Exit Continuation Lines Option	Extent Exit for DAM Core Index Entry	A/U	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind R/U/N	File Condition U1-U8	
02	F	CARDS	I														MFCU1											
03	F	PRINT	O											OV		L	PRINTER											
04	F																											
05	F																											
06	F																											
07	F																											
08	F																											
09	F																											
10	F																											
	F																											
	F																											

Different device names could be used here, depending on your system configuration and model. The device name for the Model 6 printer would be TRACTR1.

Figure 3-4. Assigning an Overflow Indicator to the Printer

Output-Format Specifications

When RPG II handles overflow, pages are advanced automatically. When you handle overflow, you must specify that forms should advance. This is done by specifying a skip to the first printing line on the page. For the end-of-month inventory report (Figure 3-1), this would be a heading on line 11. Figure 3-5 shows the correct specification for forms advancement. Remember to make a skip specification on a line conditioned by the overflow indicator (Figure 3-5). If you forget, a continuous listing will be the result.

When the printer reaches the end of a printed page, RPG II also allows you to ignore that the end of the page has been reached and continue printing. You do this by assigning an overflow indicator and never using it to condition output files. Lines will be printed from the top line to the bottom line of each page, even over the perforation. If you do not want this to happen, remember to use an overflow indicator to condition the output operations which are to be done when the end of the page is reached.

Preventing Records From Printing Over the Perforation

Suppose your program prints several detail and/or total records per program cycle as shown in Figure 3-6. In this case, the overflow indicator could be turned on:

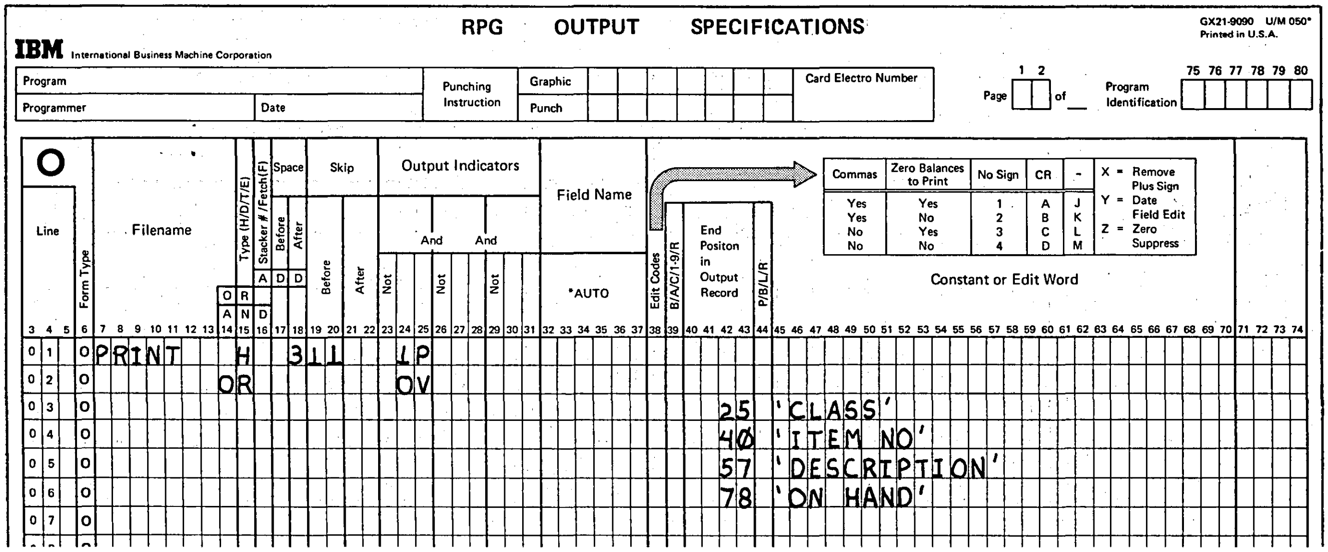
1. When the detail record is printed.

2. When any one of the total records is printed.

If overflow occurs when the detail record is printed, all total lines will also be printed before forms advance, provided a level 3 control break has occurred (L1-L3 are on). Remember the specification to skip to the next page is on the heading line conditioned by the overflow indicator. This heading line is reached only after total records are printed.

Assume that line 58 was specified as the overflow line for this program. Assume also that the detail record printed on the overflow line and that a level 3 control break occurred when the next card was read. One page of the report would look like that shown in Figure 3-7. Because all total records are printed before overflow is sensed, the last total record is printed on the fourth line of the next page. One total record was even printed over the perforation.

What can you do to eliminate this situation? You could specify the overflow line high enough on the page so that all total records would be printed on the page after the overflow line has been reached. For the report shown in Figure 3-7, the printing of total records requires 14 lines (including spacing lines). Thus for the case where a detail record is printed on the overflow line, you would have to specify line 44 as the overflow line to prevent printing past line 58.



RPG OUTPUT SPECIFICATIONS

IBM International Business Machine Corporation

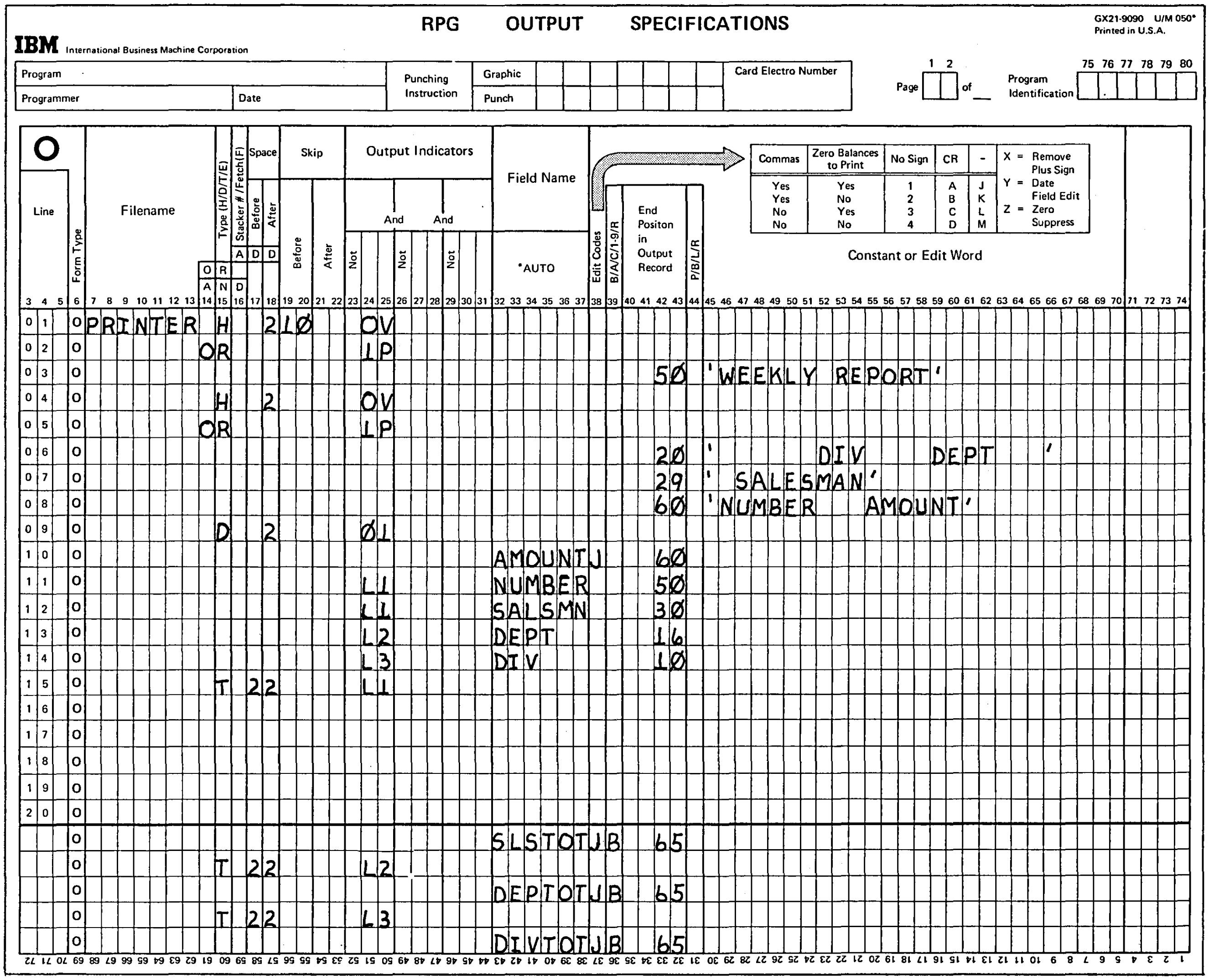
GX21-9090 U/M 050*
Printed in U.S.A.

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

```
01 OPRINT   H  311   1P
02 O       OR        OV
03 O                                   25 'CLASS'
04 O                                   40 'ITEM NO'
05 O                                   57 'DESCRIPTION'
06 O                                   78 'ON HAND'
07 O
```

Figure 3-5. Specifications for Forms Advancement



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program		Punching Instruction	Graphic
Programmer	Date		Punch

Card Electro Number

Page 1 2 of ___

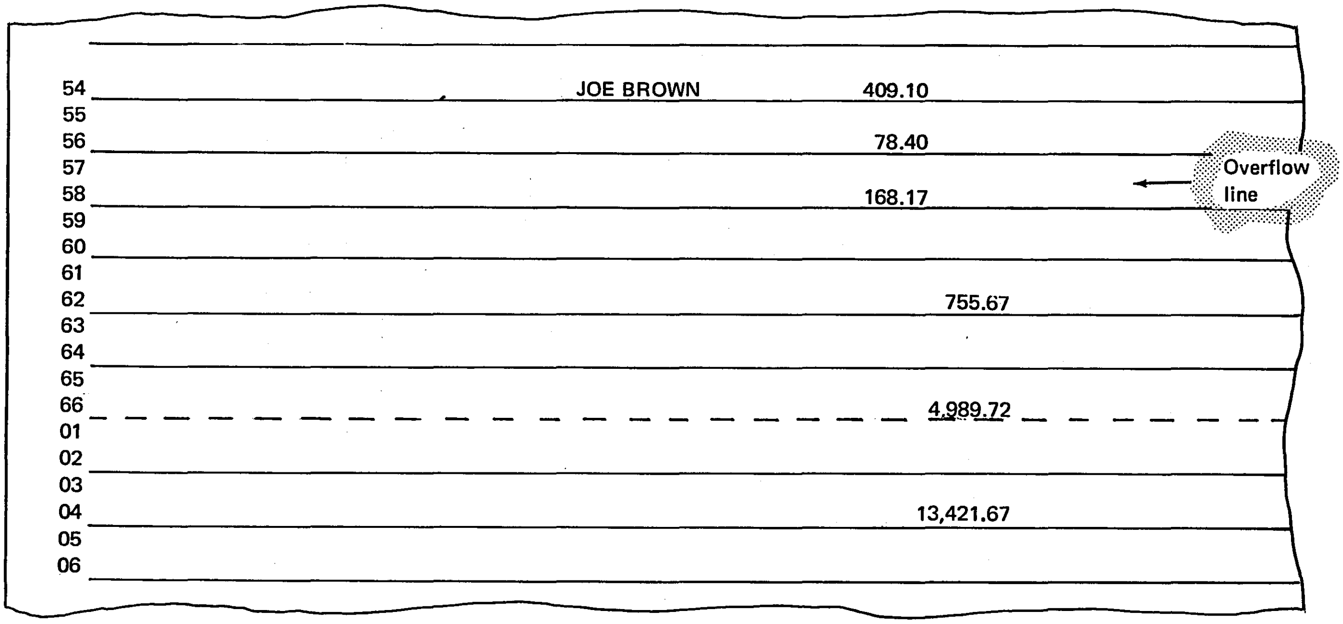
Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	AND/OR	Type (H/D/T/E)	Stacker # /Fetch (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name / *AUTO	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	PRINTER		H			2	1Ø		OV						
02	O		OR							1P						
03	O													5Ø		'WEEKLY REPORT'
04	O			H			2			OV						
05	O		OR							1P						
06	O													2Ø		' DIV DEPT '
07	O													29		' SALESMAN'
08	O													6Ø		'NUMBER AMOUNT'
09	O			D			2			Ø1						
10	O										AMOUNT	J		6Ø		
11	O									L1	NUMBER			5Ø		
12	O									L1	SALSMN			3Ø		
13	O									L2	DEPT			16		
14	O									L3	DIV			1Ø		
15	O			T		2	2			L1						
16	O															
17	O															
18	O															
19	O															
20	O															
	O										SLSTOT	J	B	65		
	O			T		2	2			L2						
	O										DEPTOT	J	B	65		
	O			T		2	2			L3						
	O										DIVTOT	J	B	65		

Figure 3-6. Several Total Records Per Cycle.



```
54                          JOE BROWN          409.10
55
56                                              78.40
57
58                                             168.17
59
60
61
62                                                         755.67
63
64
65
66 — — — — — — — — — — — — — — — — — — — —   4,989.72 — — — — — —
01
02
03
04                                                      13,421.67
05
06
```

Figure 3-7. Printing Over the Perforation

This is not the best solution, however. Suppose overflow was caused by the second total record instead of the detail record. Only one more total line would be printed before forms advanced. Much of the page is not used in this case since the last line is printed on line 50 (see Figure 3-8). As you see, this is a very uneconomical solution since much paper can be wasted.

Fetch Overflow

RPG II provides you with a better solution for preventing printing over the perforation than the one previously discussed. This solution uses the RPG II routine known as *fetch overflow.* Fetch overflow specifications allow you to alter the basic RPG II overflow logic (see *Overflow Indicator, Chapter 1*). You can cause forms to advance at the time total or detail records are printed, instead of waiting for the usual time. Figure 3-9 shows the two additional times when operations conditioned by the overflow indicator may be performed. (Remember that forms advance at this time.)

During the regular program cycle, the RPG II program tests only once to see if the overflow indicator is on; this occurs immediately after total output. By using the fetch overflow specification, you can tell the computer to check if the overflow indicator is on before it prints total or detail records. You do this by simply entering an F in column 16 of the Output-Format sheet for any detail or total record. When an F is encountered, a test is made before that line is printed.

If the overflow indicator is on when the test is made, all operations conditioned by the overflow indicator are immediately performed. These operations usually include forms advancement and the printing of headings. In order for the line to be printed, all other indicator conditions tested for on the same line as the overflow indicator must also be satisfied.

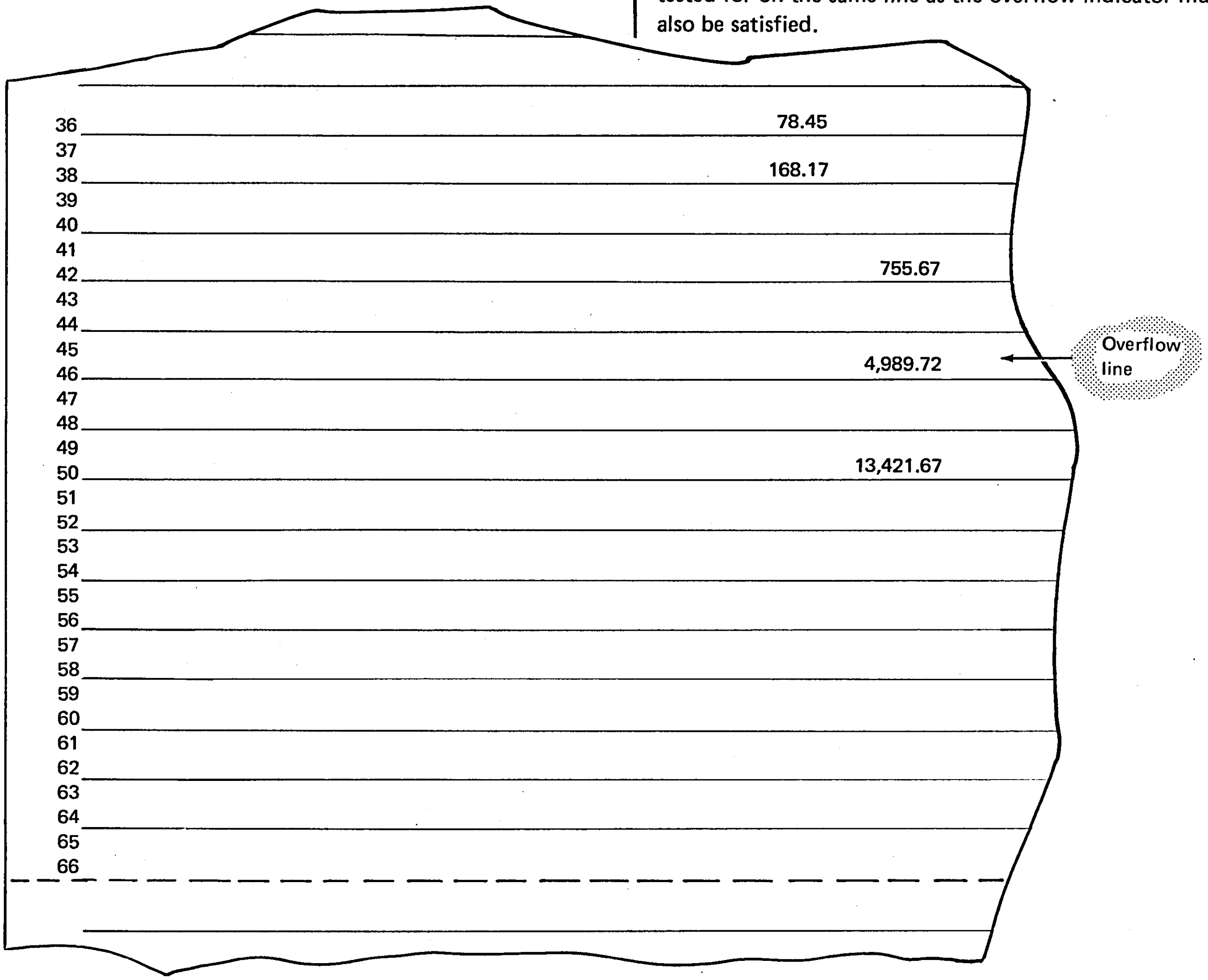


Figure 3-8. Specifying the Overflow Line High on the Page

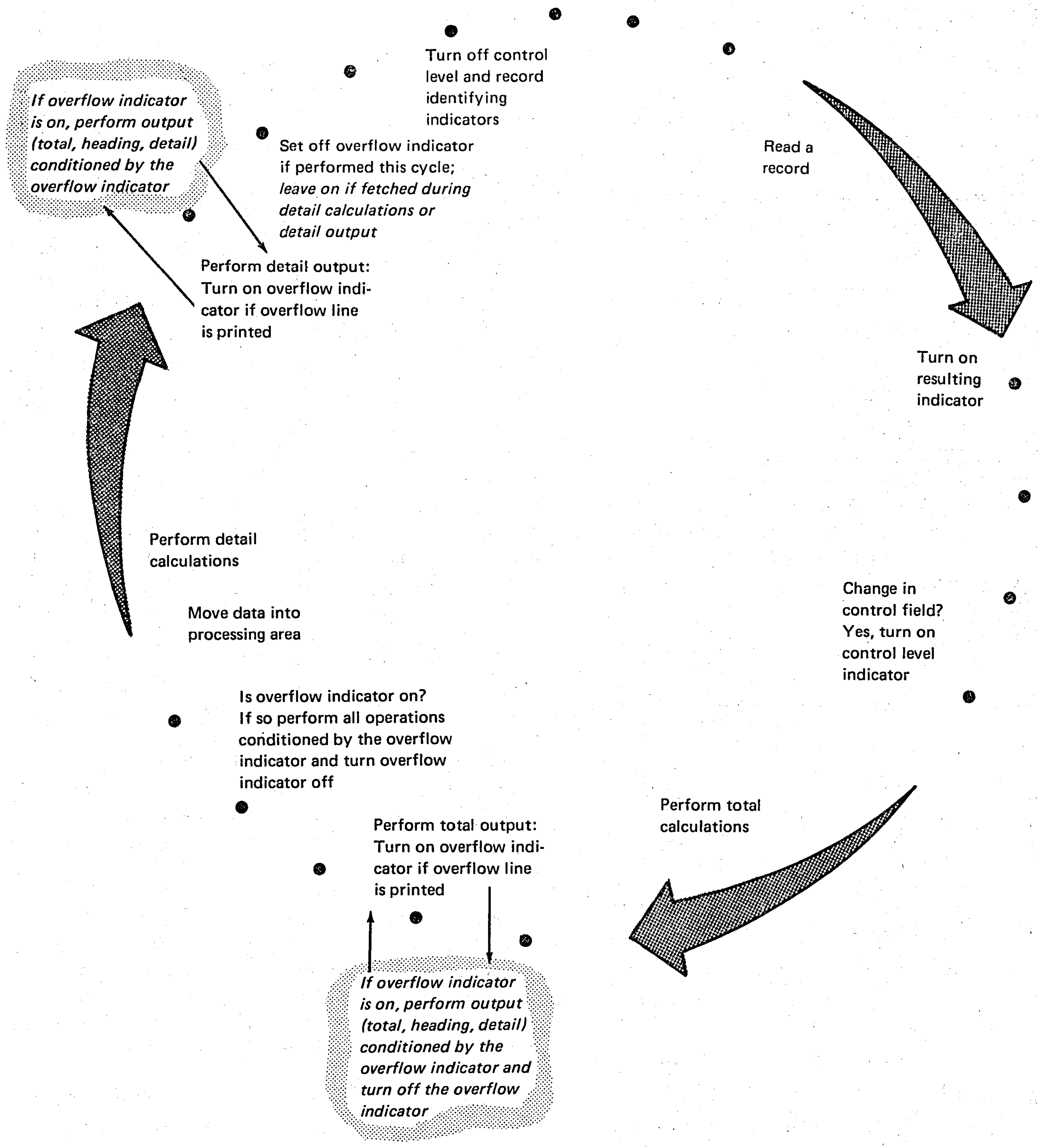

Read a record
Turn on resulting indicator
Change in control field? Yes, turn on control level indicator
Perform total calculations
Perform total output: Turn on overflow indicator if overflow line is printed
If overflow indicator is on, perform output (total, heading, detail) conditioned by the overflow indicator and turn off the overflow indicator
Is overflow indicator on? If so perform all operations conditioned by the overflow indicator and turn overflow indicator off
Move data into processing area
Perform detail calculations
Perform detail output: Turn on overflow indicator if overflow line is printed
If overflow indicator is on, perform output (total, heading, detail) conditioned by the overflow indicator
Set off overflow indicator if performed this cycle; leave on if fetched during detail calculations or detail output
Turn off control level and record identifying indicators

Figure 3-9. Logic for Fetch Overflow

Figure 3-10 shows two fetch overflow specifications (lines 07 and 09). Consider how these operations are performed. When it is time for the specification in line 07 to be done, a test is made to see if the overflow indicator is on. If it is on, the overflow routine is fetched; this causes the following operations to be performed.

1. All total lines conditioned by the overflow indicator are printed.

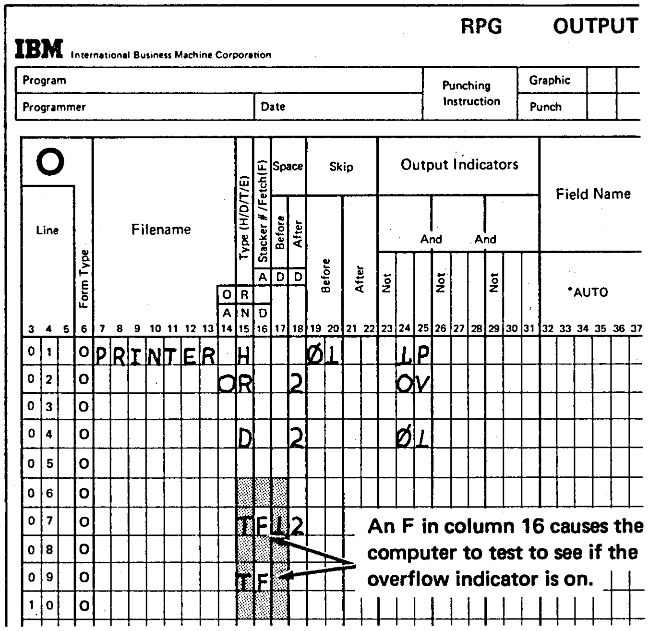
2. Forms are advanced (provided a skip to a new page has been specified in a line conditioned by the overflow indicator).

3. Heading lines conditioned by the overflow indicator are printed.

4. The overflow indicator is turned off.

5. The record specified in line 07 is printed.

Another test is made to see if the overflow indicator is on because of the specification (F) in line 09. If line 07 causes forms to advance, the overflow indicator would not be on at this time. The total record specified in specification line 09 would be printed normally.



IBM International Business Machine Corporation

RPG OUTPUT

Program		Punching Instruction	Graphic		
Programmer	Date		Punch		

Line	Form Type	Filename	AND/OR	Type (H/D/T/E)	Stacker #/Fetch(F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name
01	O	PRINTER		H				Ø1		LP	
02	O		OR				2			OV	
03	O										
04	O			D			2			Ø1	
05	O										
06	O										
07	O			T	F	1	2				
08	O										
09	O			T	F						
10	O										

An F in column 16 causes the computer to test to see if the overflow indicator is on.

Figure 3-10. Fetch Overflow Specifications

However, if the record specified in line 07 were printed on the overflow line, the overflow indicator would be on and the specification in line 09 would cause the overflow routine to be performed.

Consider again the example as shown in Figure 3-6. When the detail line was printed on the overflow line, all total lines were also printed before forms advanced. As a result, printing occurred over the perforation onto the next page (Figure 3-7).

What records should most logically be printed on an overflow page? Your answer is probably all those records that printed on or over the perforation. It would indeed be nice if all total records could be printed on the next page when a detail record was printed on the overflow line (see Figure 3-11).

If the program knew before it printed the first total record that the overflow line had been reached, forms could be advanced before the total records were printed. By specifying an F in column 16 of the first total specification, you can tell the program to check to see if the overflow indicator is on. If it is on at this time, forms will advance before total records are printed. Specifying an F in column 16 of the first total specifications will cause all total records to be printed on an overflow page.

Would an F for the first total line take care of all situations? Suppose that overflow did not occur until the first total record was printed. The remaining total lines, having no fetch overflow specification in column 16, would not cause the program to check to see if the overflow indicator was on. Thus, they would be printed on the same page (Figure 3-12). Counting the spaces, this would mean the last print line would be eight lines beyond 58 (the overflow line) or on line 66.

If this is feasible for your report, you could allow printing on line 66. If not, you could have the overflow indicator checked at the second total line. In this case, if the first total line caused the overflow indicator to turn on, the second total line would fetch the overflow routine. Thus, the last total records would be printed on the overflow page.

How can you determine on which line to place the F that will fetch the overflow routine (provided the overflow indicator is on)? You should study all possible overflow situations. By counting spaces and lines, you can calculate what would happen if overflow occurred on each detail and total line. This is essentially the method used in the previous discussion.

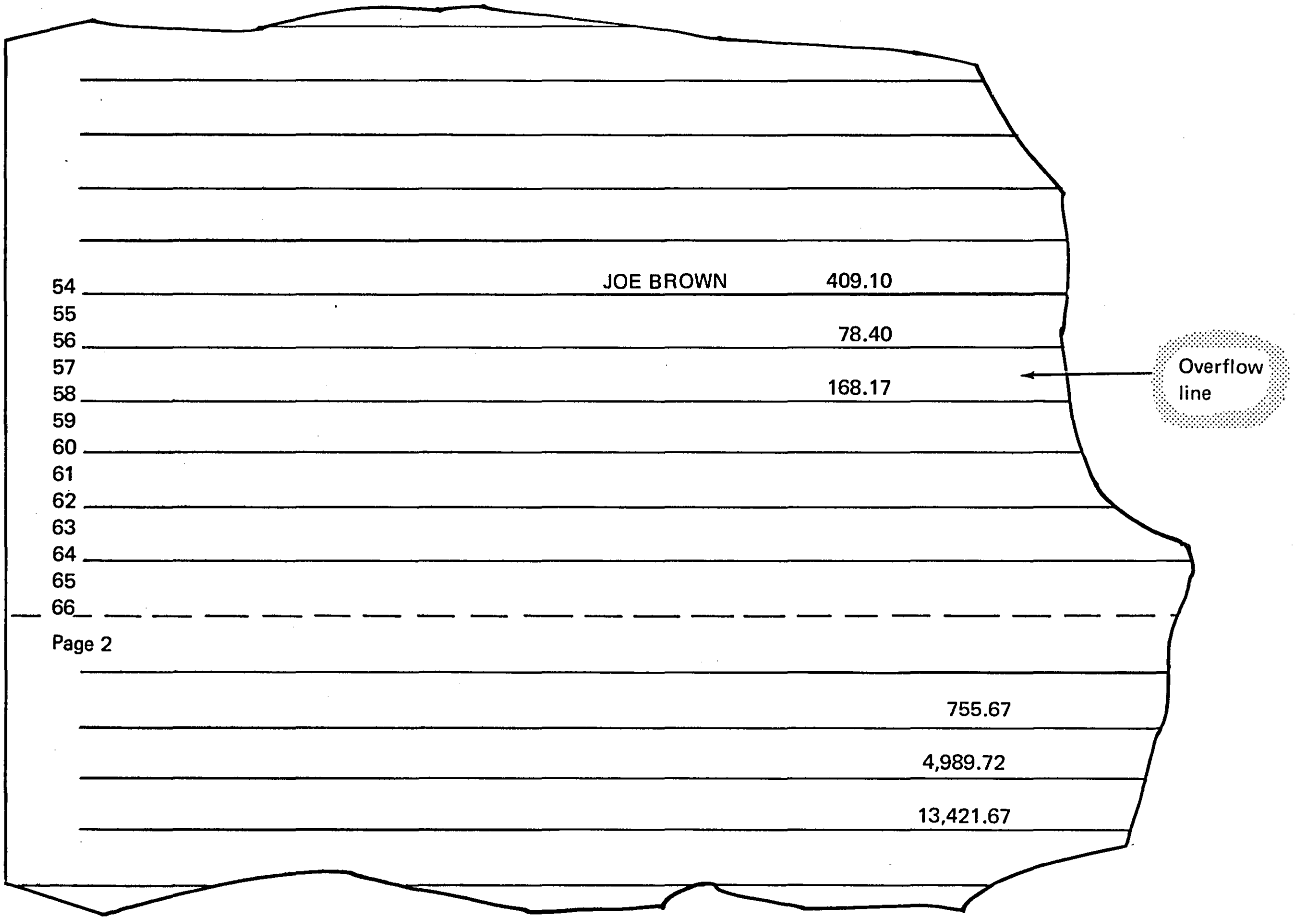


Figure 3-11. Printing Total Records on the Overflow Page

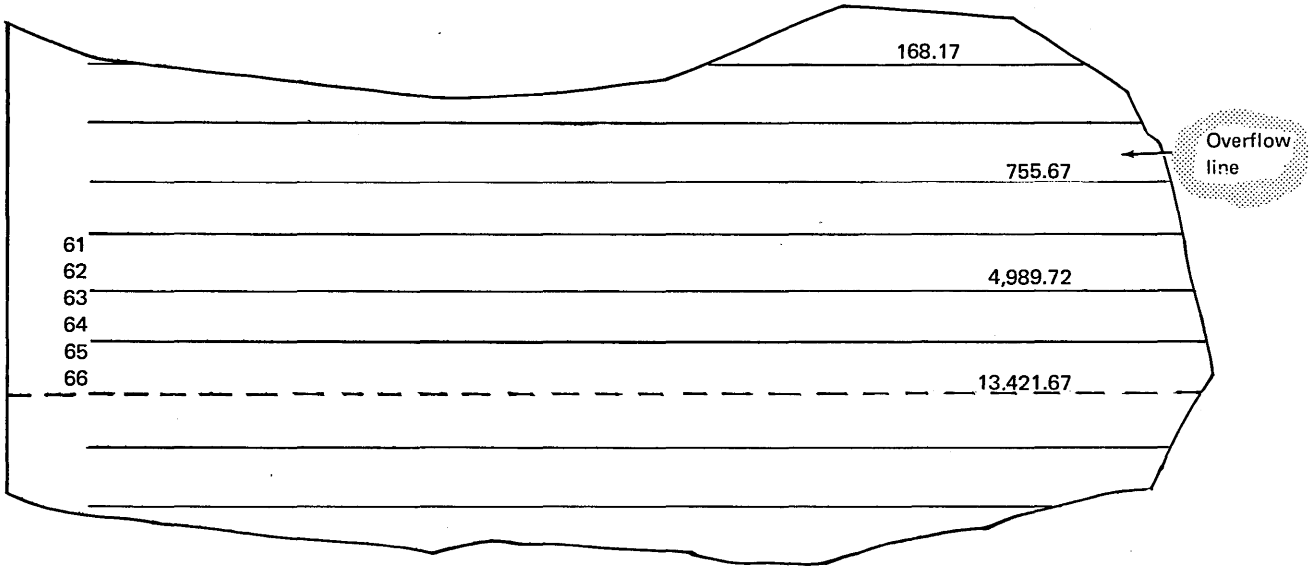


Figure 3-12. Printing on the Last Line on the Page

ALIGNING FORMS

Regardless of the type of printing forms you are using, it is always necessary to have the forms aligned so that printing is done *on* the correct line. If printing occurs above or below the line, your report looks messy and is hard to read.

How can you be sure the first line will be printed in the correct position? You can align the forms in the position you feel is correct. But you can never be sure until you try. To try the alignment, the program must be executed; a record must be printed. Suppose the forms are incorrectly aligned, and as a result, the first printing line is incorrectly positioned. In this case, you can stop the computer and realign the forms so that, hopefully, the second line will be correct. This can go on for several tries, however. In the meantime, the first few records printed on the report will have been incorrectly aligned.

RPG II has the facility to print the first line repeatedly until forms are aligned properly. This eliminates printing the first several lines of a report before correct forms alignment is attained. The use of this facility requires two specifications:

1. An output line conditioned by the 1P indicator.
2. The entry of 1 in column 41 of the control card.

When these specifications are made, the first line conditioned by the 1P indicator is printed. All processing then stops. The operator has time to reposition the forms if necessary. When this is done, the operator has the option of having the 1P line printed again or of continuing processing. This he indicates by specific settings of the console switches on the processing unit. (See halt 1P in the *IBM System/3 Models 8 and 10 Halt Guide*, GC21-7540 for a complete description of the alignment process.)

All space and skip entries specified for the 1P line are performed when forms are being aligned. This should be considered in planning for forms alignment.

If spooling printed output on the Model 15, the 1P forms alignment option will be ignored and the user can request alignment by means of the OCL PRINTER statement.

EDITING

Formatting a printed report is one way of making the report easy to read and understand. Formatting, however, concerns only the spacing and arrangement of data on the printed page. It does not concern the data itself. Data must also be readable before the report can be understood. Editing makes a field readable.

When fields are printed out according to basic specifications they appear exactly as they are inside the computer. This is shown by the following examples:

Type of Field	*Field Before Printing*	*Field After Printing*
Alphameric	JOHN T SMITH	JOHN T SMITH
Numeric	004765K	004765K

The alphameric field, when printed, is easy to read and understand, but the numeric field is confusing. How should it be read? What does the K mean? K is actually a combination of a digit and the sign of the field. But in this form, the reader would have to guess what it says (see *Character Structure in Working With Data Structure* for more information).

Editing is the means by which data is made more readable and understandable. Editing a field means punctuating it by removing the sign of the field from the rightmost digit and placing it at the end of the field, adding commas, decimal points, minus signs, dollar signs, or any other constant information.

Only numeric fields need to be edited before they are printed. Notice the difference between the following edited and unedited data taken from the same numeric amount field:

004765K	unedited data
$476.52CR	edited data

The edited amount field is certainly much more precise and understandable than the unedited field.

Methods of Editing

A field can be edited by two methods: (1) edit codes and (2) edit words. Several different codes are available. Each code edits in a slightly different way according to a predefined pattern. All, however, remove the sign of the field so that the rightmost digit will always print as a number. (See *Character Structure* in *Working With Data Structure* for more information.) The Y edit code is used for date fields only.

Figure 3-13 shows the edit pattern for all codes. Choose the code which will edit a field the way you want it to appear and enter this code in column 38 of the Output-Format sheet.

Edit Code	Commas	Decimal Point	Sign For Negative Balance			Print Out On Zero Balance *			Zero Suppress
			No Sign	CR	– (Minus)	Domestic and United Kingdom	World Trade /	World Trade J	
1	Yes	Yes	No Sign			.00 or 0	,00 or 0	0,00 or 0	Yes
2	Yes	Yes	No Sign			Blanks	Blanks	Blanks	Yes
3		Yes	No Sign			.00 or 0	,00 or 0	0,00 or 0	Yes
4		Yes	No Sign			Blanks	Blanks	Blanks	Yes
A	Yes	Yes		CR		.00 or 0	,00 or 0	0,00 or 0	Yes
B	Yes	Yes		CR		Blanks	Blanks	Blanks	Yes
C		Yes		CR		.00 or 0	,00 or 0	0,00 or 0	Yes
D		Yes		CR		Blanks	Blanks	Blanks	Yes
J	Yes	Yes			–	.00 or 0	,00 or 0	0,00 or 0	Yes
K	Yes	Yes			–	Blanks	Blanks	Blanks	Yes
L		Yes			–	.00 or 0	,00 or 0	0,00 or 0	Yes
M		Yes			–	Blanks	Blanks	Blanks	Yes
X **									
Y ***									Yes
Z									Yes

* Zero balances for the World Trade format are written in two ways, depending on the entry made in column 21 of the control card specifications.

** The X code performs no editing.

*** The Y code is used for date fields. It suppresses the leftmost zero only. The Y code edits a three to six digit field according to the following pattern:
nn/n
nn/nn
nn/nn/n
nn/nn/nn
If a data field of six digits is packed on disk and the Y edit code is used with the data field, an error will occur. To solve this problem, move the data field to another field.

Figure 3-13. Edit Codes

For example, if you wish a field called AMOUNT which has two decimal positions to be zero suppressed and punctuated with decimal points and commas (when needed) but with no sign, you choose the code that will do this. The chart in Figure 3-13 shows that two codes will accomplish this—1 and 2. If you wish a zero balance to print, you would choose the code 1. If you wanted blanks to print when the field is zero, you would use the 2 code.

Using edit codes is a convenient way of editing. However, the codes by themselves can't do everything you might want to do.

Punctuating With a Dollar Sign

Suppose you wanted a dollar sign to be printed on the report for the AMOUNT field. An edit code won't put the dollar sign there. You will have to specify this in addition to the edit code you are using.

According to the printer spacing chart (Figure 3-14), the AMOUNT field is six characters long. It begins in column 70 and ends in column 75. However, the minus sign would extend the amount field to column 76, as shown in Figure 3-15 and Figure 3-16. The dollar sign, if printed, should be in column 69. Line 11 in Figure 3-15 shows the specification for editing the AMOUNT field by the edit code J. This code is used so that negative values will print with a minus sign (-) following the field. The dollar sign can be specified as a constant ending in column 69 and must be specified in line 12, the line following the edit code.

Depending upon its contents, the AMOUNT field, when printed out, could look like any of the following (where N is any number):

```
$NNN.NN
$ NN.NN
$  N.NN
$   .NN
```

The blanks between the first digit and the dollar sign are the result of zero suppression. The dollar sign remains in position 69; this is known as a *fixed dollar sign.*

Often, it is desirable, as in writing checks, to eliminate these empty spaces. There are two ways of doing this: (1) moving the dollar sign so that it is always next to the first digit; and (2) filling the spaces with asterisks.

Instead of having blanks between the dollar sign and the first digit, you can cause the dollar sign to print next to the first digit. In this case, the amount field would look like one of the following:

```
$NNN.NN
 $NN.NN
  $N.NN
   $.NN
```

A dollar sign that changes positions is known as a *floating dollar sign.* It is specified by placing the entry '$' in columns 45-47 on the same specification line as the edit code (see Figure 3-16). Remember that the fixed dollar sign was specified by placing $ in columns 45-47 of the line *following* the edit code (Figure 3-15).

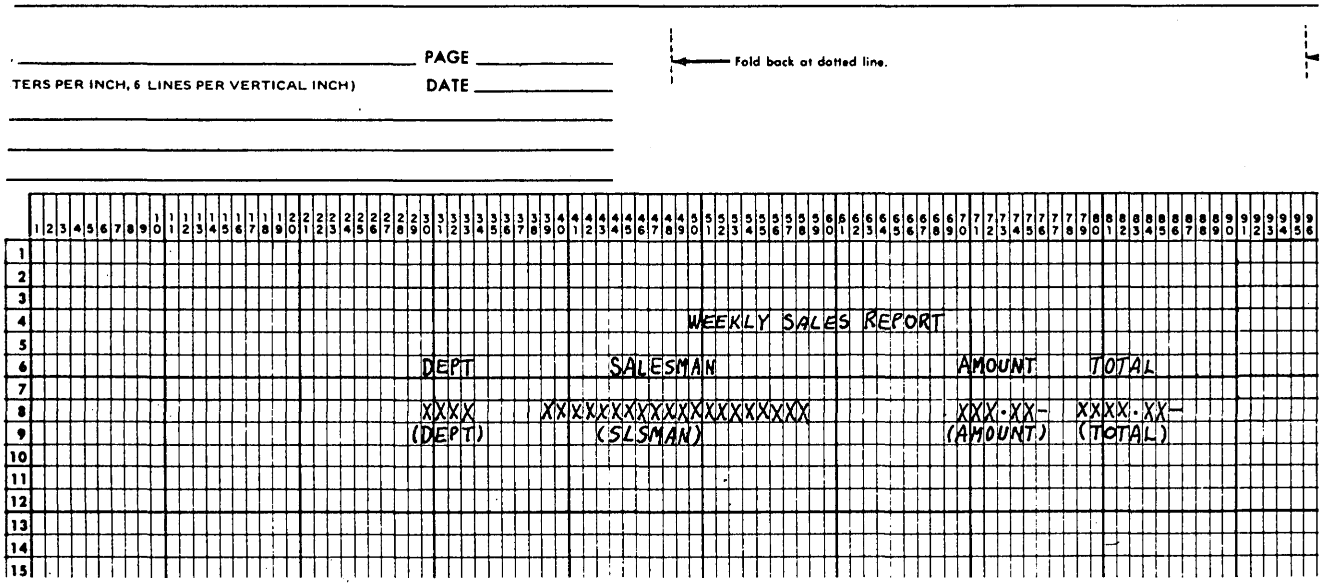
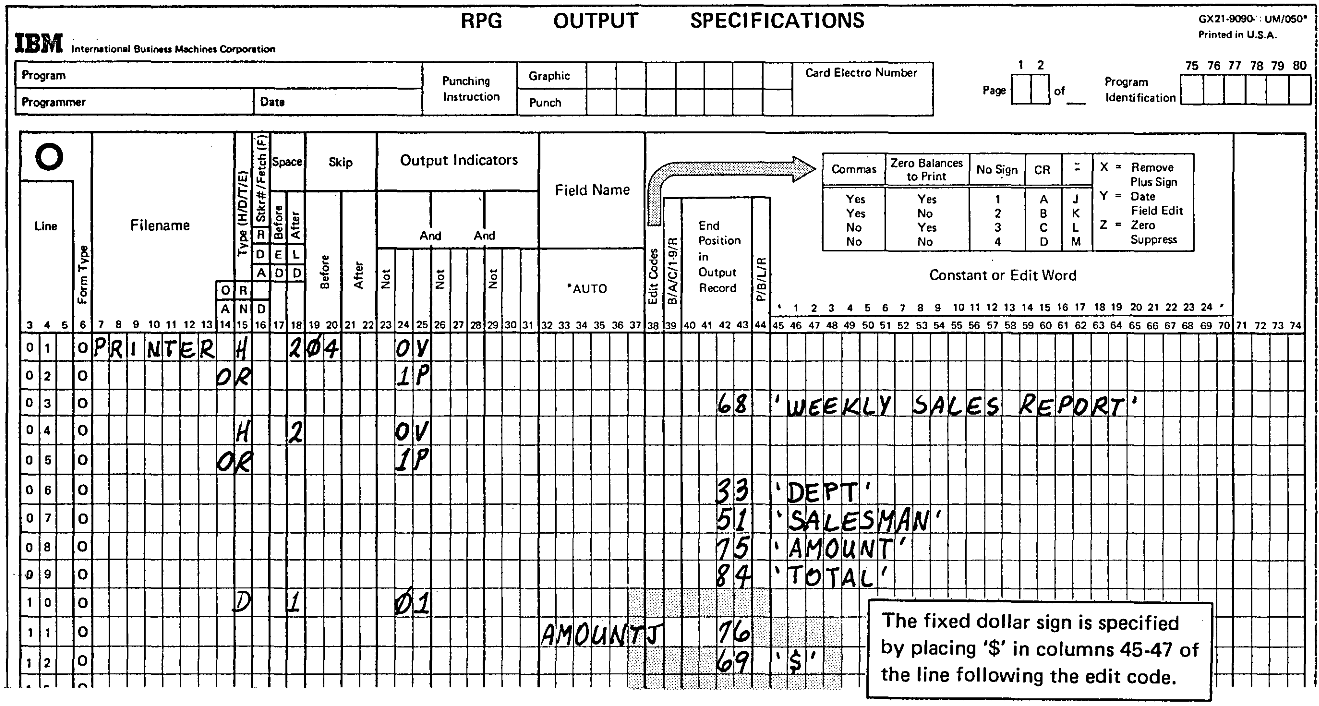


Figure 3-14. Printer Spacing Chart (Editing)



IBM International Business Machines Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 UM/050 Printed in U.S.A.

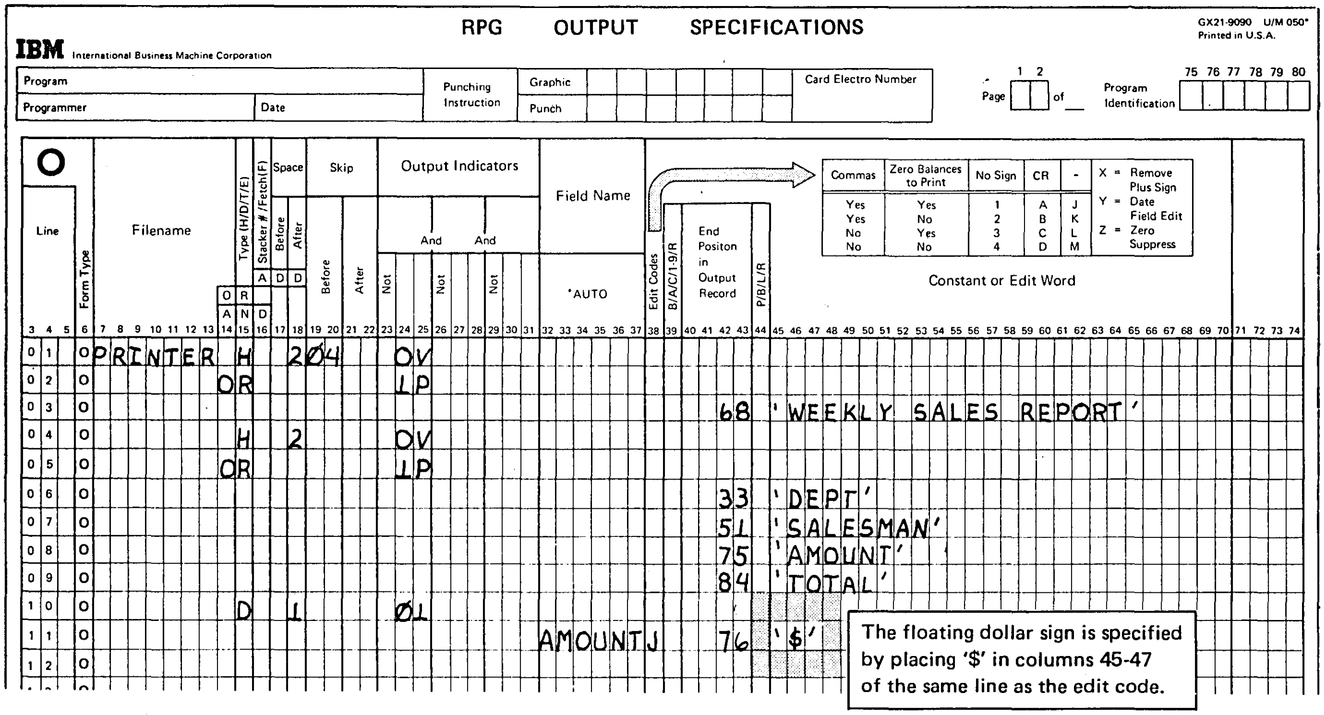
Program ____ Programmer ____ Date ____ Punching Instruction: Graphic / Punch ____ Card Electro Number ____ Page of ____ Program Identification ____

Commas	Zero Balances to Print	No Sign	CR	-	X = Remove Plus Sign Y = Date Field Edit Z = Zero Suppress
Yes	Yes	1	A	J	
Yes	No	2	B	K	
No	Yes	3	C	L	
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	PRINTER	H	2	04	OV				
02	O	OR				1P				
03	O								68	'WEEKLY SALES REPORT'
04	O		H	2		OV				
05	O	OR				1P				
06	O								33	'DEPT'
07	O								51	'SALESMAN'
08	O								75	'AMOUNT'
09	O								84	'TOTAL'
10	O		D	1		01				
11	O						AMOUNT	J	76	
12	O								69	'$'

The fixed dollar sign is specified by placing '$' in columns 45-47 of the line following the edit code.

Figure 3-15. Fixed Dollar Sign



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050 Printed in U.S.A.

Program ____ Programmer ____ Date ____ Punching Instruction: Graphic / Punch ____ Card Electro Number ____ Page of ____ Program Identification ____

Commas	Zero Balances to Print	No Sign	CR	-	X = Remove Plus Sign Y = Date Field Edit Z = Zero Suppress
Yes	Yes	1	A	J	
Yes	No	2	B	K	
No	Yes	3	C	L	
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	End Positon in Output Record	Constant or Edit Word
01	O	PRINTER	H	2	04	OV				
02	O	OR				LP				
03	O								68	'WEEKLY SALES REPORT'
04	O		H	2		OV				
05	O	OR				LP				
06	O								33	'DEPT'
07	O								51	'SALESMAN'
08	O								75	'AMOUNT'
09	O								84	'TOTAL'
10	O		D	1		01				
11	O						AMOUNT	J	76	'$'
12	O									

The floating dollar sign is specified by placing '$' in columns 45-47 of the same line as the edit code.

Figure 3-16. Floating Dollar Sign

Punctuating With Asterisks

To indicate that asterisks should fill the spaces caused by suppression of leading zeros, place the entry '*' in columns 45-47 of the same specification line as the edit code. If a dollar sign is to appear before the asterisk, it must be specified on the next line. With the specifications shown in Figure 3-17, the AMOUNT field, depending upon its contents, could look like any of the following:

$NNN.NN
$*NN.NN
$* *N.NN
$* * *.NN

Refer to the *IBM System/3 RPG II Reference Manual*, SC21-7504 for a more detailed explanation and further considerations.

Punctuating With Dashes

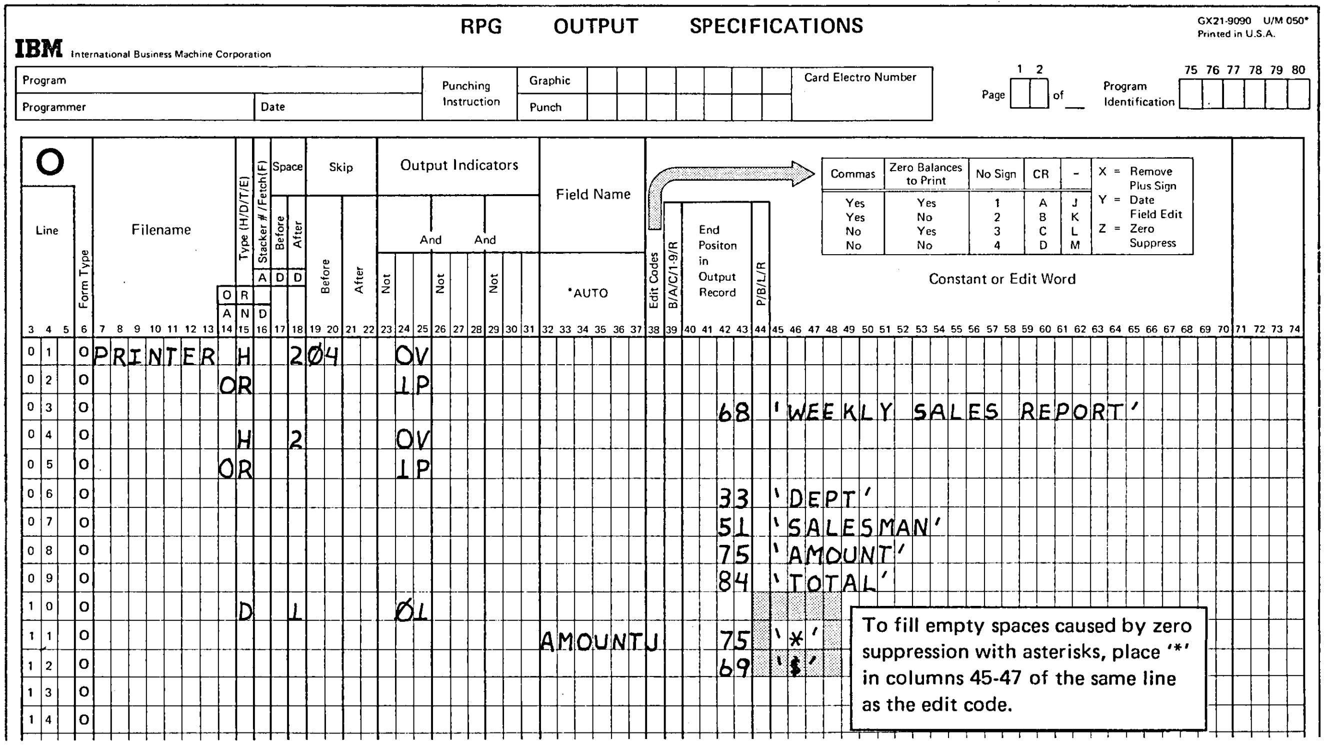
What code would you use for the following job? A report listing all employee names, addresses, telephone numbers, and social security numbers, is desired. A file is kept on all employees. Each record includes one employee name, address, telephone number, and social security number among other things.

The telephone number and social security number are entered in the record as one continuous number with no dashes. Remember that fields will print out exactly as they are recorded. Thus the telephone number will appear as 2820804. This is rather hard to read. 282-0804 is much better. How will you get the dash to appear?

A dash is a type of punctuation. So some type of editing must be done. Can you find a code that will edit this way? No, there is none available.

For this case, you will have to set up your own editing pattern. An *edit word* gives a pattern for punctuation. For the phone number you need three digits, a dash, and then four digits. You specify the edit word in columns 45-70 of the Output-Format sheet. The word, like any constant, must be enclosed in quotes. Figure 3-18, line 04, shows how the telephone number field is edited: three blanks, a dash, and four blanks.

Unless the social security field is also edited, it will print out in one long string of numbers, such as 472446357. Form the edit word to make the social security number read: 472-44-6357. It should look like the edit word shown in Figure 3-19, line 05. Notice that a leading 0 (zero) is included in the edit word in addition to the number of places required for the data. This prevents zero suppression when the SOCSEC field is edited.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

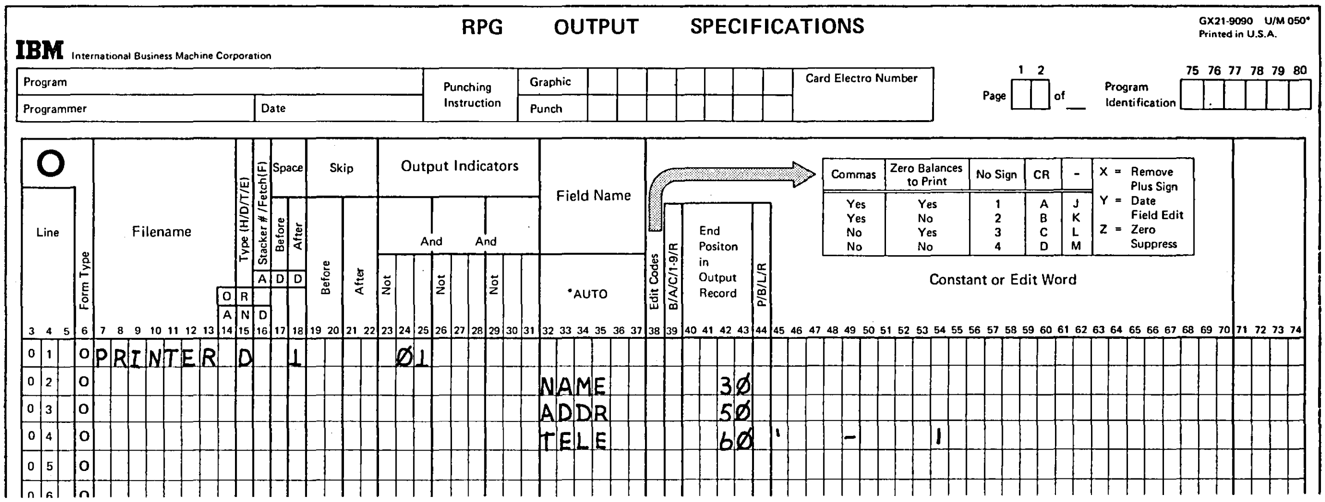
Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Space	Skip	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	PRINTER	H	2	Ø4	OV				
02	O		OR			1P				
03	O								68	'WEEKLY SALES REPORT'
04	O		H	2		OV				
05	O		OR			1P				
06	O								33	'DEPT'
07	O								51	'SALESMAN'
08	O								75	'AMOUNT'
09	O								84	'TOTAL'
10	O		D	1		Ø1				
11	O						AMOUNT	J	75	'*'
12	O								69	'$'
13	O									
14	O									

To fill empty spaces caused by zero suppression with asterisks, place '*' in columns 45-47 of the same line as the edit code.

Figure 3-17. Punctuating with Asterisks



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program

Programmer | Date

Punching Instruction | Graphic | Punch

Card Electro Number

Page 1 2 of

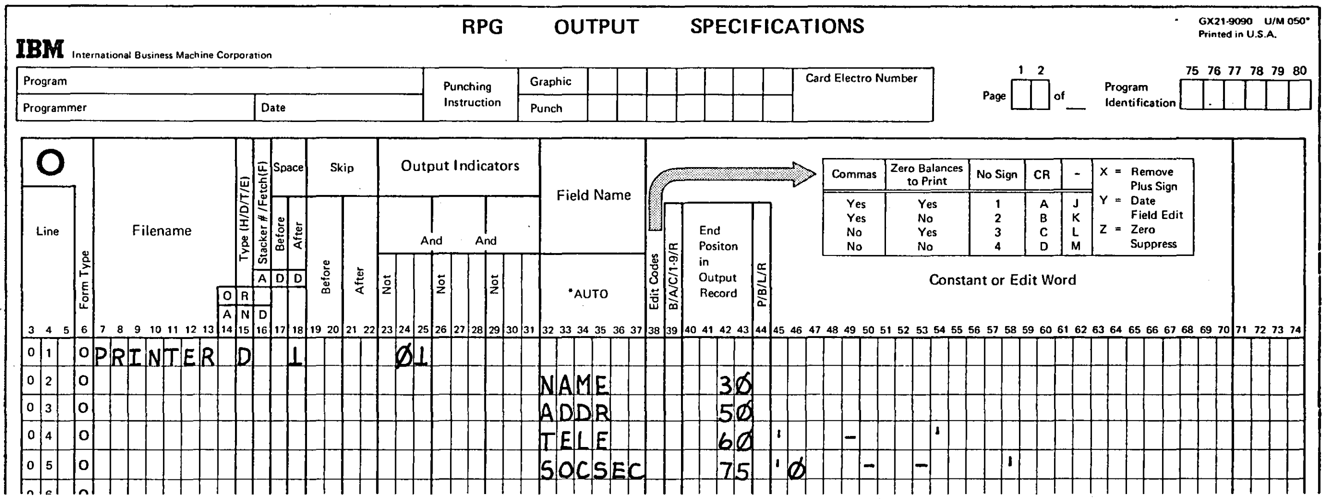
Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

```
Line  Form  Filename  Type(H/D/T/E)  Space  Skip  Output Indicators  Field Name  End Position  Constant or Edit Word
3 4 5 6 7 ...
01 OPRINTER D  1     Ø1
02 O                         NAME      3Ø
03 O                         ADDR      5Ø
04 O                         TELE      6Ø '   -    '
05 O
06 O
```

Figure 3-18. Edit Word for Telephone Number



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program

Programmer | Date

Punching Instruction | Graphic | Punch

Card Electro Number

Page 1 2 of

Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

```
Line  Form  Filename  Type(H/D/T/E)  Space  Skip  Output Indicators  Field Name  End Position  Constant or Edit Word
3 4 5 6 7 ...
01 OPRINTER D  1     Ø1
02 O                         NAME      3Ø
03 O                         ADDR      5Ø
04 O                         TELE      6Ø '   -    '
05 O                         SOCSEC    75 'Ø   -  -    '
06 O
```

Figure 3-19. Edit Word for Social Security Number

Punctuating by Leaving Blanks

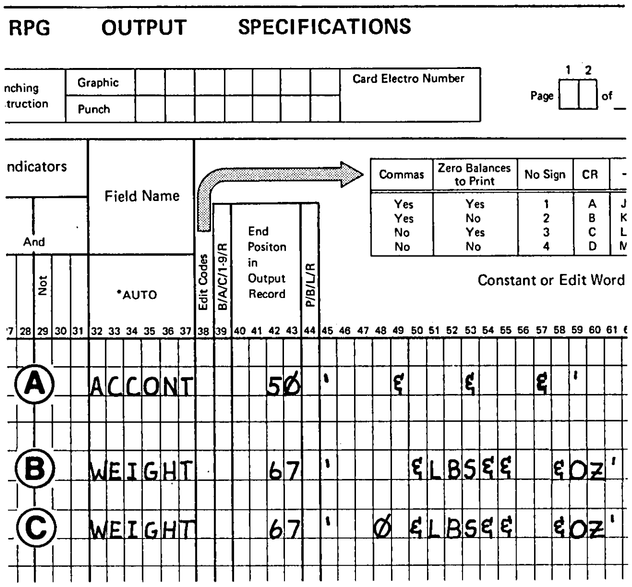
A company has rather long account numbers. To make them easier to read and handle, blanks are left after every third digit when the number is printed.

Is there a code that will insert blanks? No, again you have to specify your own edit word. In an edit word, blanks indicate where the digits go and *ampersands* indicate where blanks will go. The account field consists of ten digits. The edit word shown in Figure 3-20, insert A, will put blanks after every three digits.

Punctuating by Adding Constant Information

For shipping purposes, the catalog department of a department store must know the weight of every item it sells. The weight in pounds and ounces is recorded in a 6-digit field. The last two digits are ounces, the first four digits are pounds. When printed out on a report, the constants LBS and OZ must be inserted. Otherwise, the data would not be understandable. Again to do this, an edit word is needed because no edit code will insert LBS and OZ.

The printed field is shown on the printer spacing chart in Figure 3-21. One space is needed between the digit and LBS and between the digit and OZ. Two spaces separate pounds from ounces. Remember in the edit word, blanks are specified for digits, ampersands are specified for blanks; and constant information is inserted where desired. The edit word should be as long as the field shown on the printer spacing chart. Figure 3-20, insert B, shows the correct edit pattern for this field.

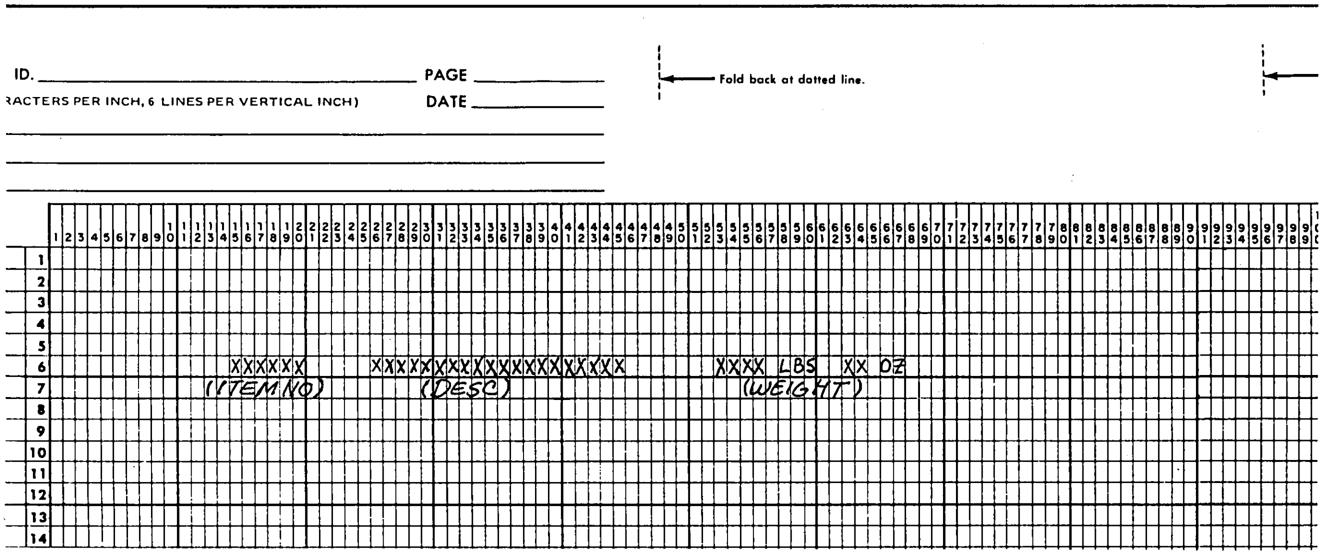

RPG OUTPUT SPECIFICATIONS
nching
truction
Graphic
Punch
Card Electro Number
1 2
Page
of
ndicators
And
Not
Field Name
*AUTO
Edit Codes
B/A/C/1-9/R
End Positon in Output Record
P/B/L/R
Commas
Zero Balances to Print
No Sign
CR
Yes Yes 1 A J
Yes No 2 B K
No Yes 3 C L
No No 4 D M
Constant or Edit Word
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61

A ACCONT 5Ø ' & & & '

B WEIGHT 67 ' &LBS&& &OZ'

C WEIGHT 67 ' Ø &LBS&& &OZ'

Figure 3-20. Edit Words


ID.
PAGE
RACTERS PER INCH, 6 LINES PER VERTICAL INCH)
DATE
Fold back at dotted line.
1 2 3 4 5 6 7 8 9 10 ... 100
1
2
3
4
5
6
7
8
9
10
11
12
13
14

XXXXXX XXXXXXXXXXXXXXXXXXXXX XXXX LBS XX OZ
(ITEM NO) (DESC) (WEIGHT)

Figure 3-21. Print Chart (Weight Field)

Edit words can do all kinds of editing for you. They can even be set up to do the same thing as the edit codes. All you have to do is *show* where the commas, decimal points, credit signs, etc. should appear. A zero is used to indicate where zero suppression stops. For example in Figure 3-20, insert C, the 0 shows where zero suppression is to end in the WEIGHT field.

Edit codes are a faster and more convenient way of editing than edit words. Therefore, edit words are normally used only when edit codes alone cannot accomplish the job.

Editing and End Position

When specifying end positions for fields which are to be edited, either by edit words or edit codes, be sure to allow enough room for the edited field. If the field to be edited is six characters long on the input record, do not allow only six positions for it on the printed report. By the time the field is edited, it may contain many more characters than six. For example, the WEIGHT field which is to be punctuated with the constants LBS and OZ is only a 6-character field on the input card. But when printed out after editing it requires 15 spaces. Always specify an end position on the Output-Format sheet (columns 40-43) that takes into account the length of the *edited* field.

USING *PLACE TO PRINT DUPLICATE INFORMATION

Using *PLACE, you can tell the RPG II compiler to print duplicate information. When you specify *PLACE on the Output-Format sheet, the fields listed above it will be printed in a different position on the same line. This eliminates much duplicate coding.

For example, assume that your distribution firm prepares invoices on their data processing system. The invoice (Figure 3-22) sent to each customer consists of two parts: one part the customer keeps, the other he tears off and sends along with his payment. Many fields are common to both parts of the invoice. For example, NAME and CUSTNO (customer number) are printed on the first line of each part. All fields in the fourth line of the report, except for the description (DESC) fields, and all fields in the total line are found in both parts of the invoice. The second part is almost a duplicate of the first.

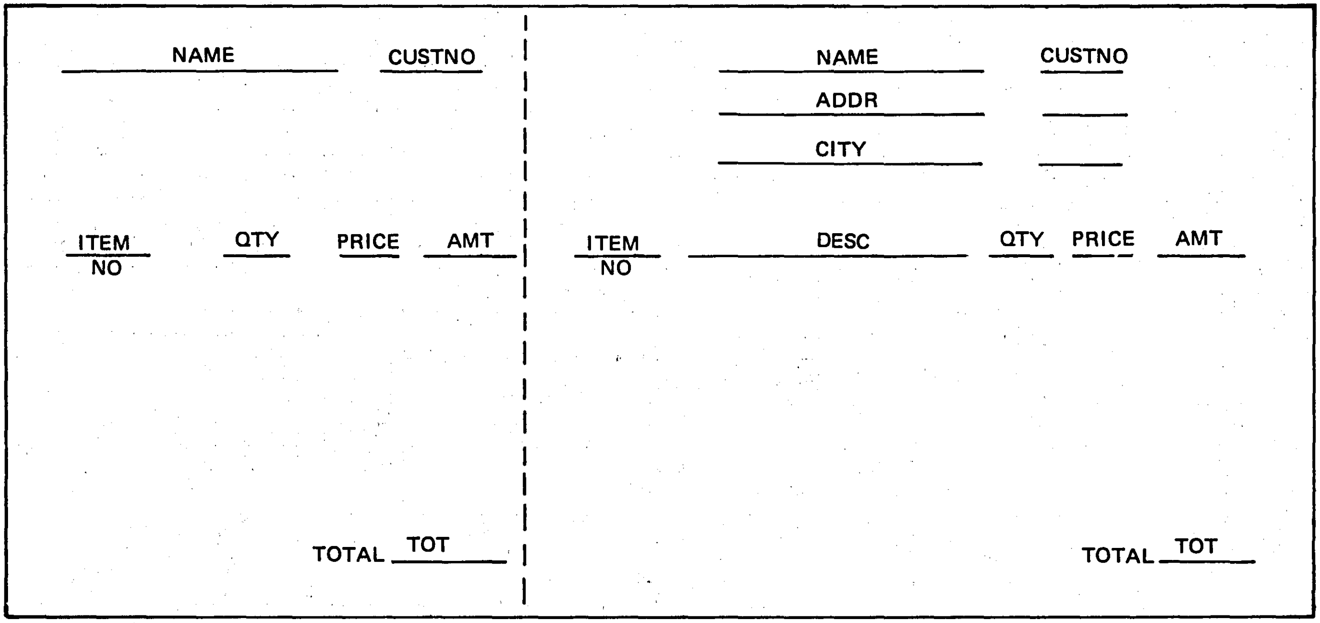


Figure 3-22. Invoice Form

Figure 3-23 shows the printer spacing chart for the invoice. What output-format specifications would you write to print fields twice on the same line? You could define the field and give the end position for it each time you wanted to print the field. Figure 3-24 shows the coding necessary using this method. There is an easier way to do this, however. This is through the use of *PLACE.

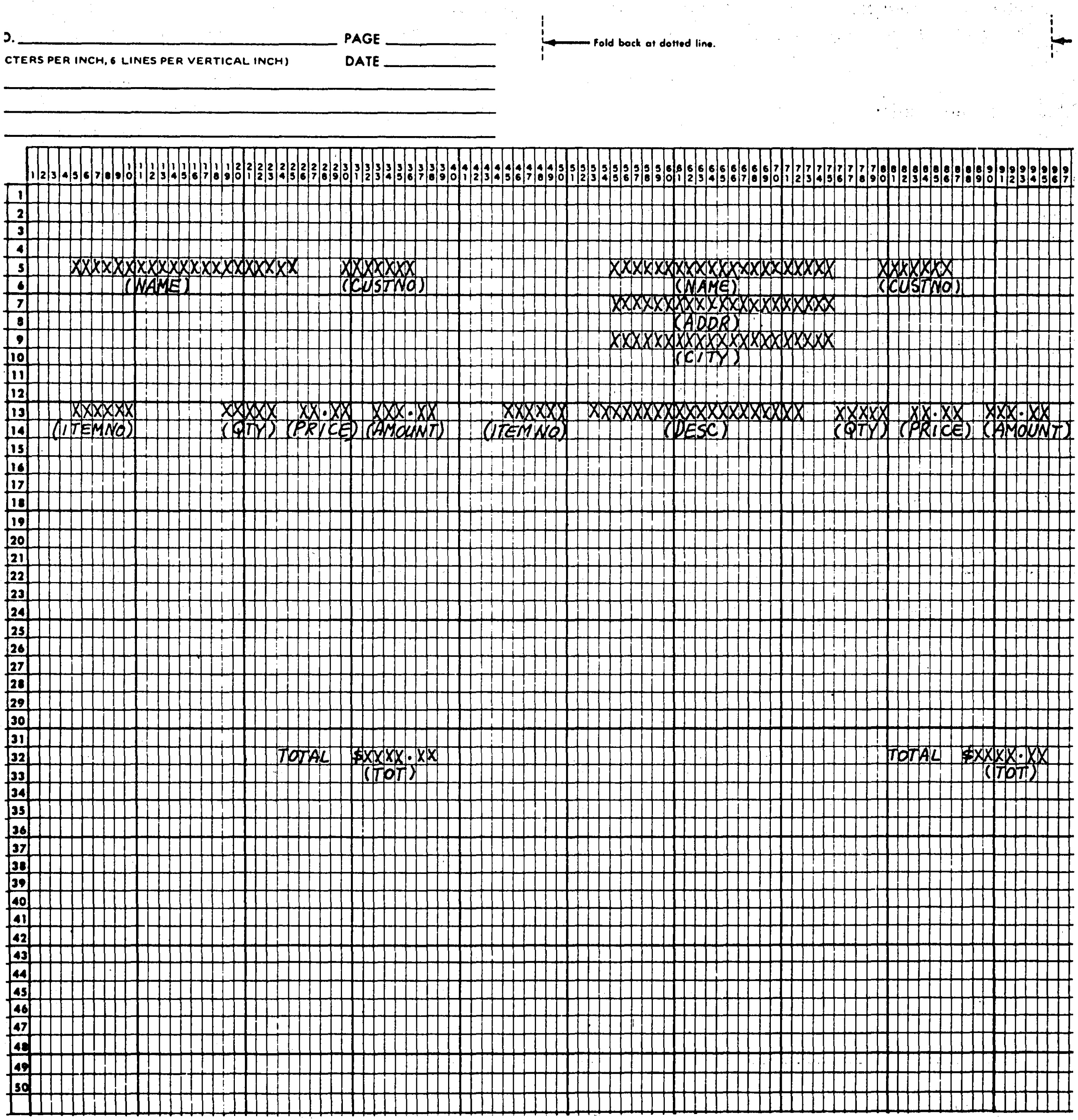
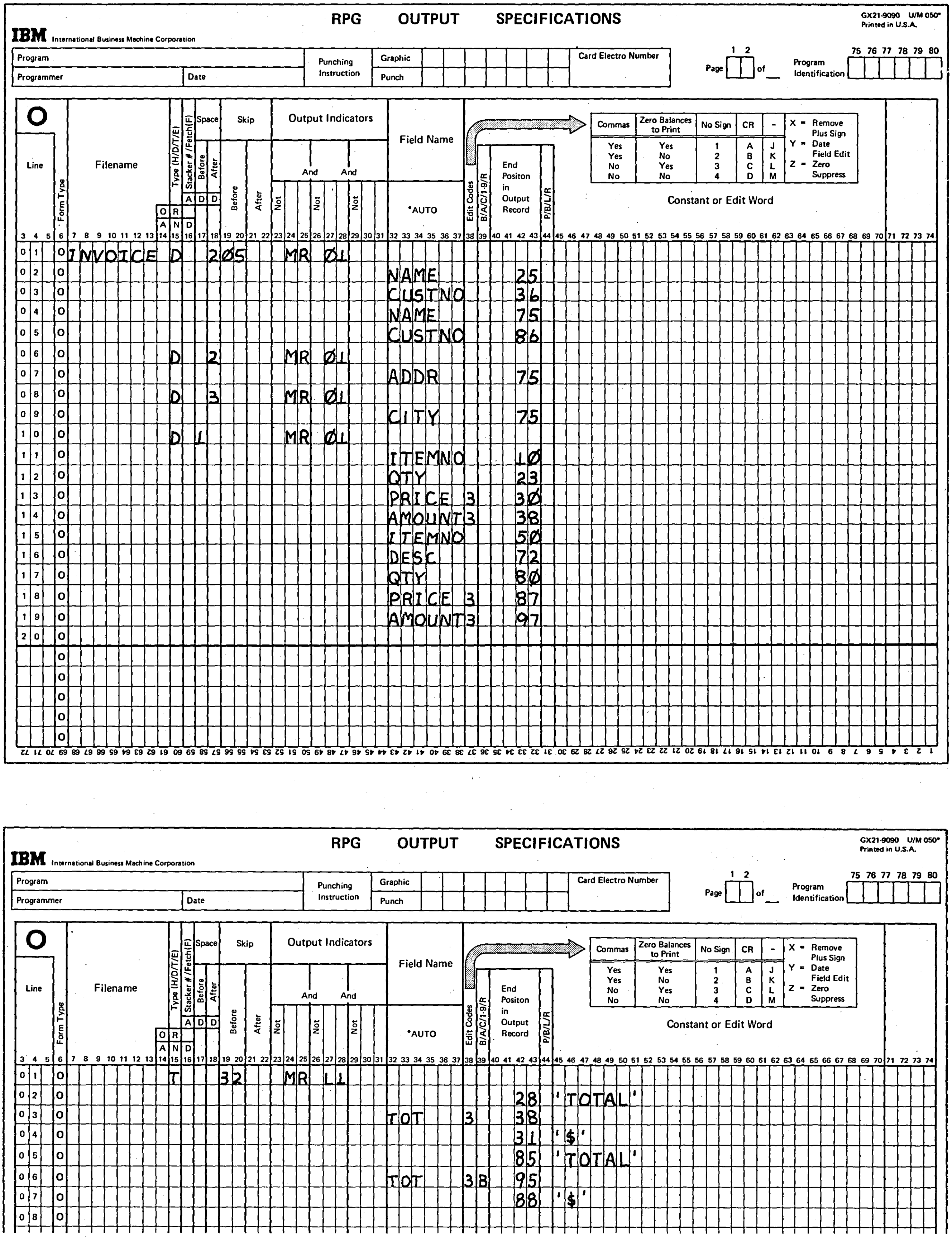


Figure 3-23. Printer Spacing Chart for Invoice



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Punching Instruction | Graphic | Card Electro Number | Page 1 2 □□ of __ | Program Identification 75 76 77 78 79 80
Programmer | Date | Punch

Commas	Zero Balances to Print	No Sign	CR	-	X = Remove Plus Sign Y = Date Field Edit Z = Zero Suppress
Yes	Yes	1	A	J	
Yes	No	2	B	K	
No	Yes	3	C	L	
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Stacker # / Fetch (F)	Space Before	Space After	Skip Before	Skip After	Not	Output Indicators	Not	And	Not	And	Field Name / *AUTO	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	INVOICE	D			2	Ø5			MR		Ø1								
02	O														NAME			25		
03	O														CUSTNO			36		
04	O														NAME			75		
05	O														CUSTNO			86		
06	O		D			2				MR		Ø1								
07	O														ADDR			75		
08	O		D			3				MR		Ø1								
09	O														CITY			75		
10	O		D		1					MR		Ø1								
11	O														ITEMNO			1Ø		
12	O														QTY			23		
13	O														PRICE	3		3Ø		
14	O														AMOUNT	3		38		
15	O														ITEMNO			5Ø		
16	O														DESC			72		
17	O														QTY			8Ø		
18	O														PRICE	3		87		
19	O														AMOUNT	3		97		
20	O																			
	O																			
	O																			
	O																			
	O																			
	O																			

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Punching Instruction | Graphic | Card Electro Number | Page 1 2 □□ of __ | Program Identification 75 76 77 78 79 80
Programmer | Date | Punch

Commas	Zero Balances to Print	No Sign	CR	-	X = Remove Plus Sign Y = Date Field Edit Z = Zero Suppress
Yes	Yes	1	A	J	
Yes	No	2	B	K	
No	Yes	3	C	L	
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Stacker # / Fetch (F)	Space Before	Space After	Skip Before	Skip After	Not	Output Indicators	Not	And	Not	And	Field Name / *AUTO	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O		T				32			MR		L1								
02	O																	28		'TOTAL'
03	O														TOT	3		38		
04	O																	31		'$'
05	O																	85		'TOTAL'
06	O														TOT	3	B	95		
07	O																	88		'$'
08	O																			

Figure 3-24. Output-Format Specifications for Invoice (Coding Each Field Twice)

Specifications for Using *PLACE

*PLACE is a special RPG II function which can be used to accomplish duplicate printing with less coding. To the RPG II compiler the specification *PLACE means: Duplicate that part of the line which has been specified and place the duplicated information in a different position on the same line. *PLACE means a special function is to be performed. You should not use this specification as a field name, since the RPG II compiler will assume you want the preceding field duplicated. When using *PLACE you first define, for each record, all the fields which are to be duplicated. Give the end position for each field as you normally do. Then enter the word *PLACE on the line below the fields which are to be duplicated. Figure 3-25 shows the entries for the first detail line of the invoice.

The compiler does not know where to print unless you specify an end position on the *PLACE entry. In Figure 3-25, the end position given for the *PLACE entry was 86.

The *PLACE specification duplicates not only letters but also blank spaces. It will duplicate all the characters (including blanks) from position 1 to the end position specified for a field. These duplicated characters are then placed so that they end in the end position specified for the *PLACE entry.

When specifying an end position for the *PLACE entry, you must know exactly where you wish the fields to print. You must also consider the amount of space needed for the printing of *all* characters to be duplicated. Always specify an end position which allows room for the printing of duplicated fields.

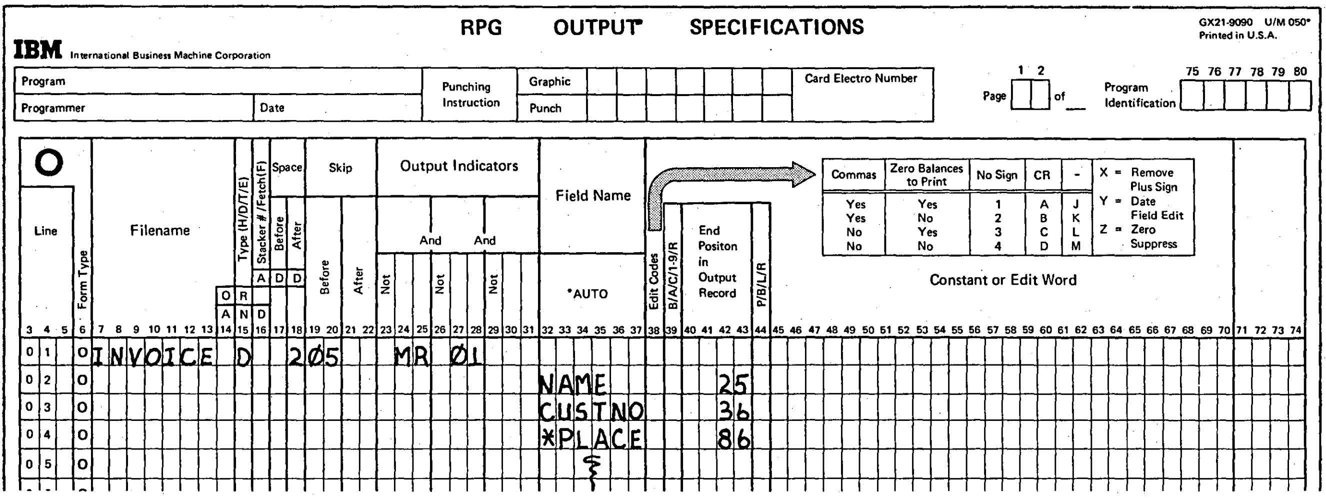
A *PLACE specification must not be conditioned by indicators in columns 23-31. *PLACE is automatically conditioned by the same indicators that condition the field or fields to be repeated.

Formation of Print Lines

When System/3 performs printer output, a whole line is printed at once, regardless of how many fields are in that line. Before printing, the whole line is moved to an area of storage exactly as it is to be printed. Data is placed in this storage area one field at a time.

The sequence in which data enters the storage area depends on the sequence that field names are specified on the RPG II Output-Format sheet. The first field recorded on the Output-Format sheet is entered first, then the second, etc. Each field is inserted into the storage area according to its end-position entry on the Output-Format sheet. If you have made conflicting entries in your specifications (for example, one field overlapping another) the last field mentioned is the one that will print in its entirety.

*PLACE operates in the same way as normal field names. The operations associated with *PLACE are performed in the sequence *PLACE is specified on the Output-Format sheet in relation to other output entries.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page □□ of __
Program Identification □□□□□□

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Output Indicators	Field Name	End Position in Output Record
01	O	INVOICE	D	2	Ø5	MR	Ø1		
02	O							NAME	25
03	O							CUSTNO	36
04	O							*PLACE	86
05	O								

Figure 3-25. Output-Format Specifications for First Line of Invoice (Using *PLACE to Print Fields Twice)

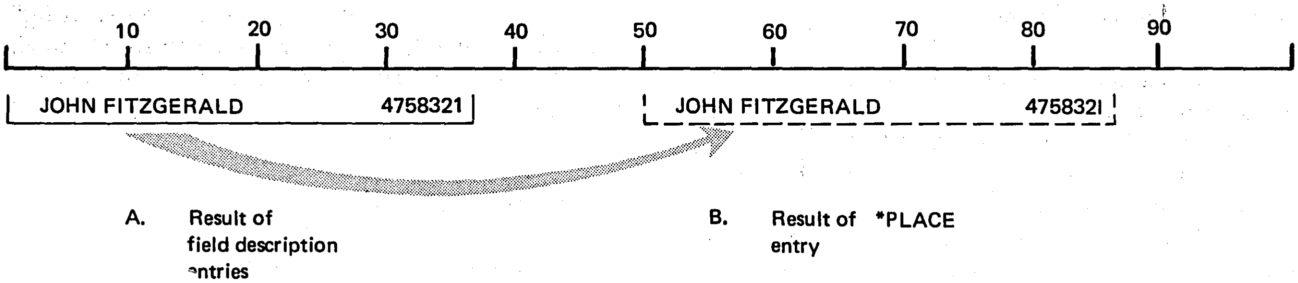


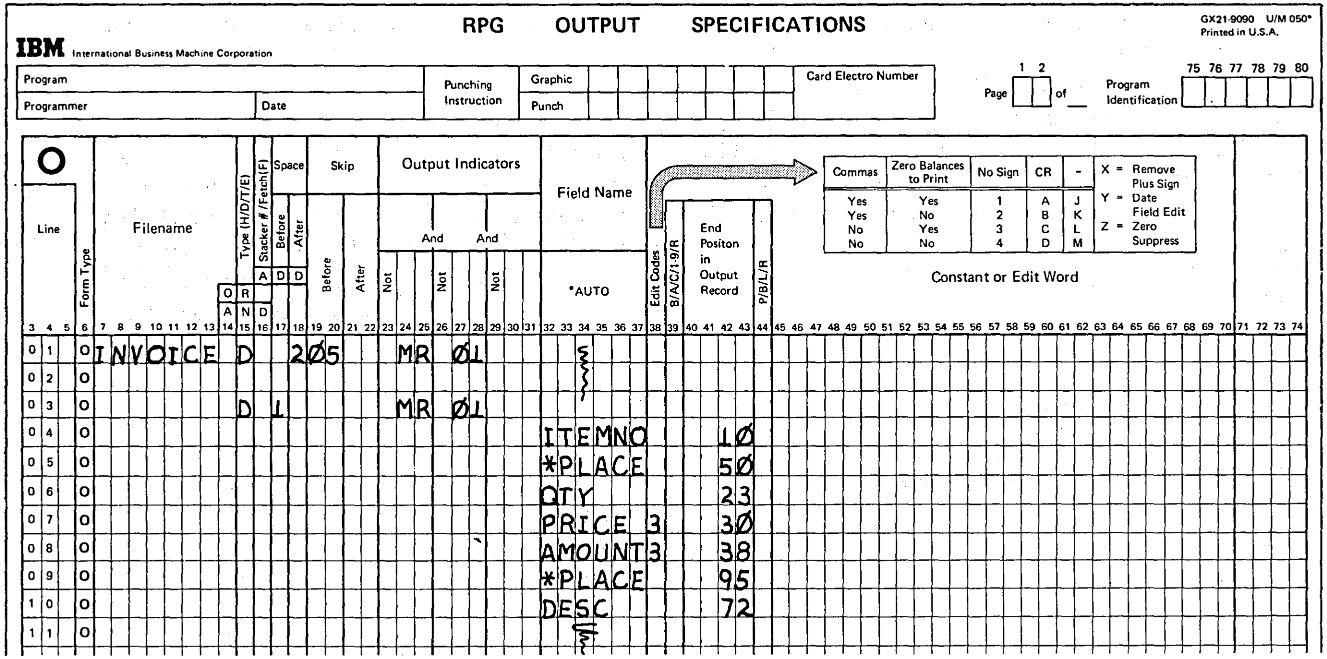
Figure 3-26. Line Formation (First Line of Invoice)

Follow the formation of the first line to be printed on the invoice. According to the specifications in Figure 3-25, the NAME field ends in position 25 and CUSTNO in 36. The first part of the line is completed with these specifications (Figure 3-26, insert A). Because of the way lines are formed, the end position for the *PLACE entry must be at least two times the higher end position specified for a field that is to be duplicated. This ensures that the last field mentioned will not overlap the field preceding. In this case the same fields are to be printed again on the second part of the same line. Since the end position was 36, the second part of the same line must end at least in position 72 (two times higher than the end position for the field to be duplicated). It is decided they are to end in position 86. The second part of the line is formed by the *PLACE entry (Figure 3-26, insert B).

Using Different Spacing for Duplicated Fields

The second and third lines of the invoice do not have fields to be duplicated. However, the fourth line of the invoice requires that all fields be duplicated. Notice that different spacing is required for the duplicated fields because a field called DESC must be inserted between ITEMNO and QTY.

Figure 3-27 shows correction specifications. You want to start with ITEMNO since it is the first fieιd. ITEMNO is specified as usual; the end position is given. Then *PLACE is specified with the correct end position, 50 in this case. These specifications cause the line to look like that in Figure 3-28, insert A.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page of
Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Constant or Edit Word

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	End Position in Output Record
01	O	INVOICE	D		2	Ø5		MR Ø1			
02	O										
03	O		D	1				MR Ø1			
04	O								ITEMNO		1Ø
05	O								*PLACE		5Ø
06	O								QTY		23
07	O								PRICE	3	3Ø
08	O								AMOUNT	3	38
09	O								*PLACE		95
10	O								DESC		72
11	O										

Figure 3-27. Specifications for Fourth Line of Invoice

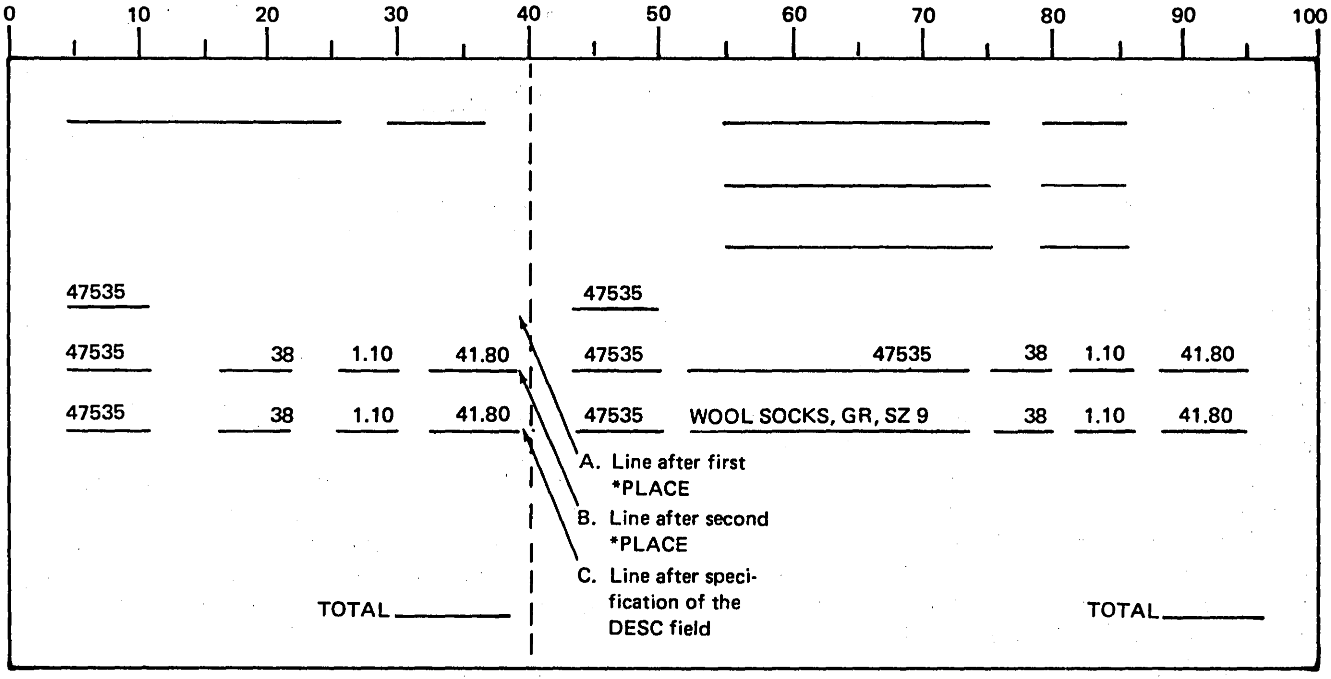


Figure 3-28. Fourth Printed Line

Now, the remaining three fields are specified and an end position is given for each. *PLACE is entered after them to signify that the above three fields should be duplicated. Remember that when fields are duplicated, all information from position 1 to the highest end position specified for a field is used. In this case, positions 1 through 38 are duplicated and placed so that they end in position 95.

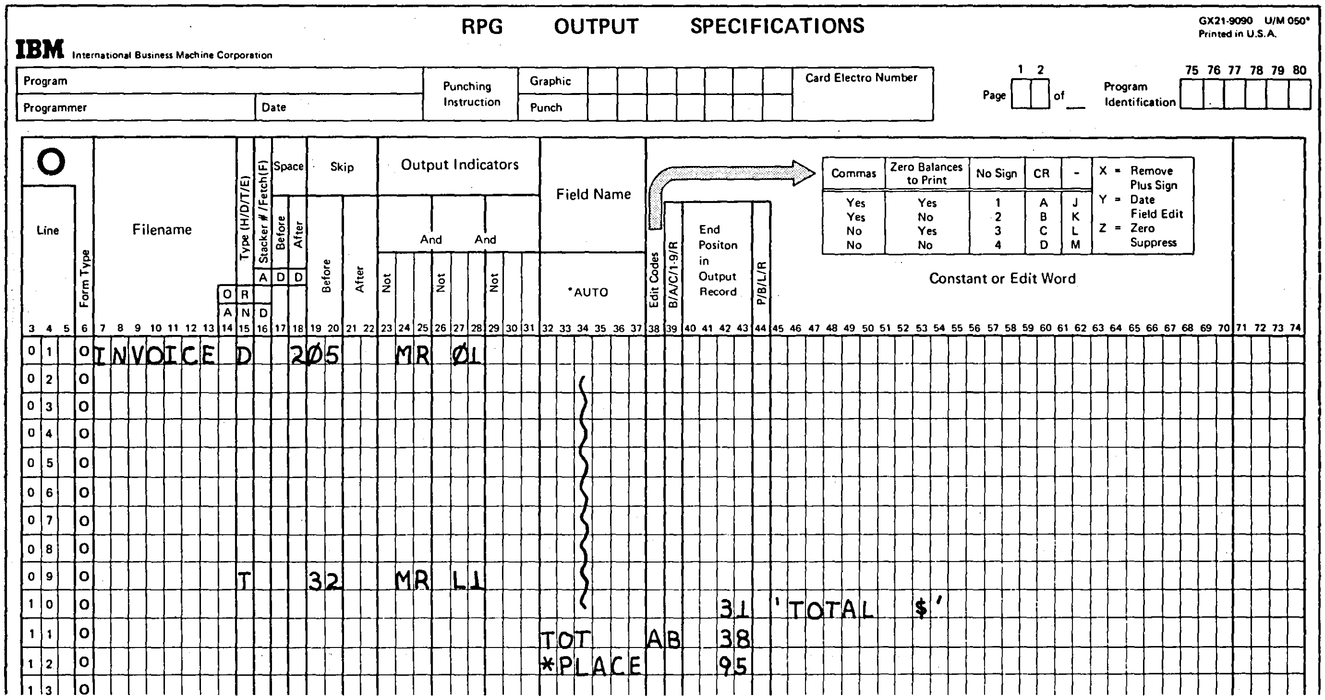
QTY, PRICE, and AMOUNT are in positions 1 through 38, but ITEMNO is also there since it ends in position 10. Thus, all four fields are duplicated and placed so that they end in 96. Figure 3-28, insert B shows resulting formation of the line. ITEMNO now appears three times, once in the DESC field area where it should not be.

In this example, we can specify the field DESC to end in position 75. It will overlay the unwanted ITEMNO field and thus get rid of it. Figure 3-28, insert C shows the line as it will be printed.

For each job you do using *PLACE, you will have to calculate exactly what happens when lines are formed.

Duplicating Constants

*PLACE can duplicate constants as well as fields. The same specifications are used for both. Figure 3-29 shows the specifications for the last line of the invoice. In this case *PLACE duplicates a field and two constants. As you can see, using *PLACE eliminates duplicate coding.


RPG OUTPUT SPECIFICATIONS
GX21-9090 U/M 050*
Printed in U.S.A.
IBM International Business Machine Corporation
Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page
of
Program Identification
O
Line
Form Type
Filename
Type (H/D/T/E)
Stacker #/Fetch(F)
Space
Before
After
Skip
Before
After
Output Indicators
And
And
Not
Not
Not
Field Name
*AUTO
Edit Codes
B/A/C/1-9/R
End Positon in Output Record
P/B/L/R
Constant or Edit Word
Commas
Zero Balances to Print
No Sign
CR
-
X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

```
  01 OINVOICE D  2Ø5   MR Ø1
  02 O
  03 O
  04 O
  05 O
  06 O
  07 O
  08 O
  09 O        T   32   MR L1
  10 O                                   31 'TOTAL  $'
  11 O                         TOT   AB  38
  12 O                         *PLACE    95
  13 O
```

Figure 3-29. Using *PLACE to Duplicate Constants (Last Line of Invoice)

Printing a Field Several Times on the Same Line

*PLACE can be used to print the same field several times in the line. All you have to do is enter *PLACE along with an end position for each time you want the fields duplicated. If you want the field duplicated twice, you need two *PLACE entries.

Assume that periodically a store prepares mailing labels for each customer who has an account with them. They use the labels when they send out special advertisements. The mailing label has only name, address, and zip code on it. Since the label has to be only a few inches wide, the manager found he could print three labels side by side on his 120-print position printer (Figure 3-30).

You can see that each field needs to be printed three times on each line. In the examples discussed so far, *PLACE was used to duplicate fields only once.

Figure 3-31 shows the specifications for the first line. NAME needs to be entered three times per line. The original field specification prints it one time: the two *PLACE entries cause it to be printed two more times.

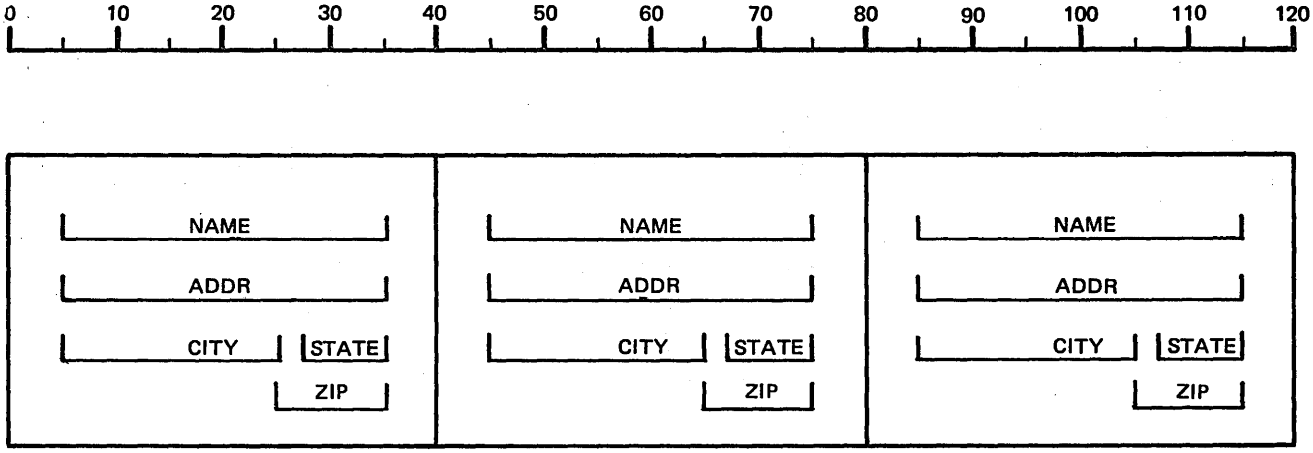
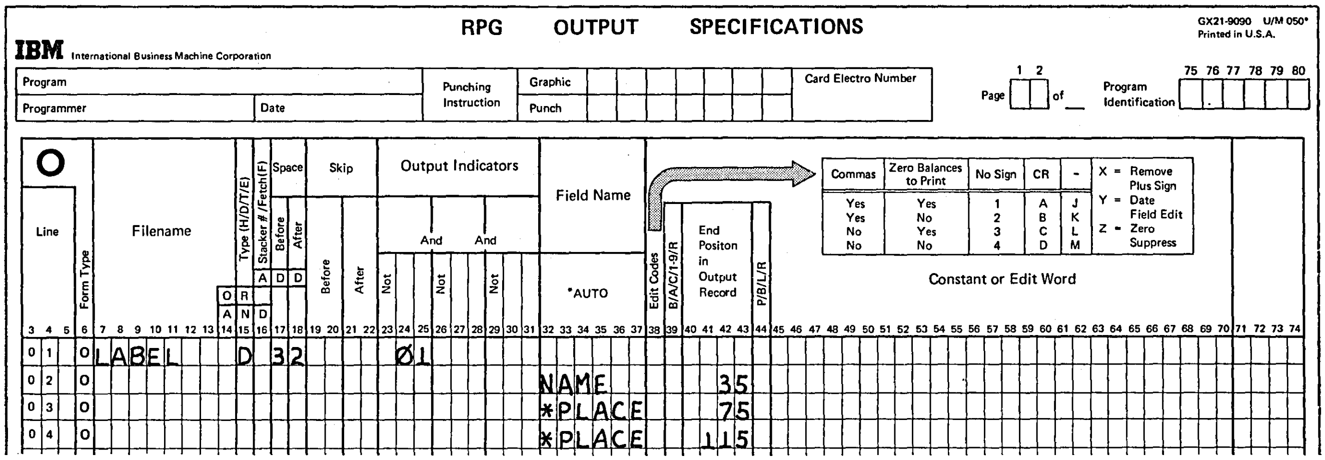


Figure 3-30. Mailing Labels



IBM International Business Machines Corporation

RPG OUTPUT SPECIFICATIONS

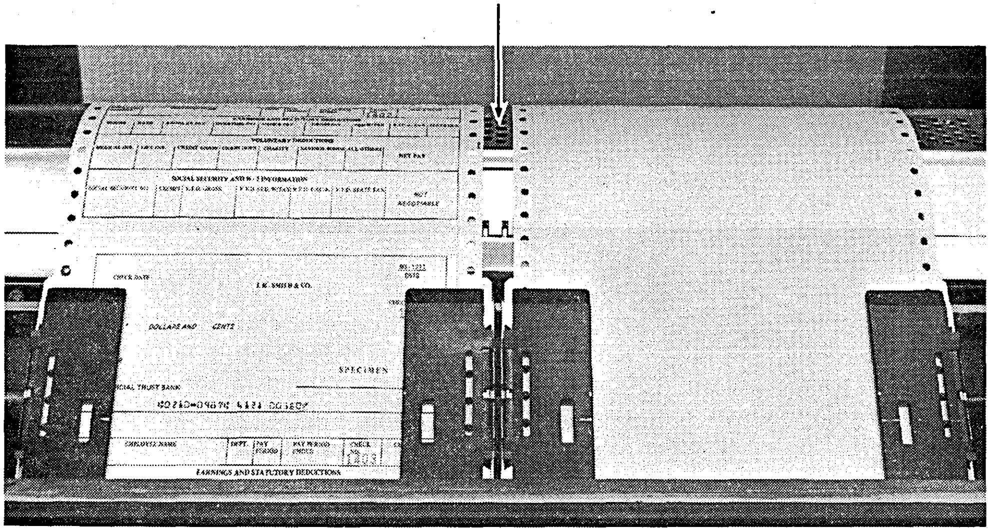
GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Output Indicators	Field Name	End Position in Output Record
01	O	LABEL	D	3	2	Ø1		
02	O						NAME	35
03	O						*PLACE	75
04	O						*PLACE	115

Figure 3-31. Using *PLACE for Producing Mailing Labels

USING TWO PRINTER OUTPUT FILES IN ONE PROGRAM

Two System/3 printers, the IBM 5203 Printer on the Model 10 Card and Disk systems and the IBM 2222 Printer on the Model 6 allow you to produce two separate printer output files on the same printer in a program (Figures 3-32 and 3-33). The forms can be narrower than standard forms and are often special forms such as checks or invoices.



A minimum of 17 print positions, between the left carriage tractor and the right carriage tractor, are not available for printing.

Figure 3-32. 5203 Printer with the Dual Feed Carriage Feature

53958A

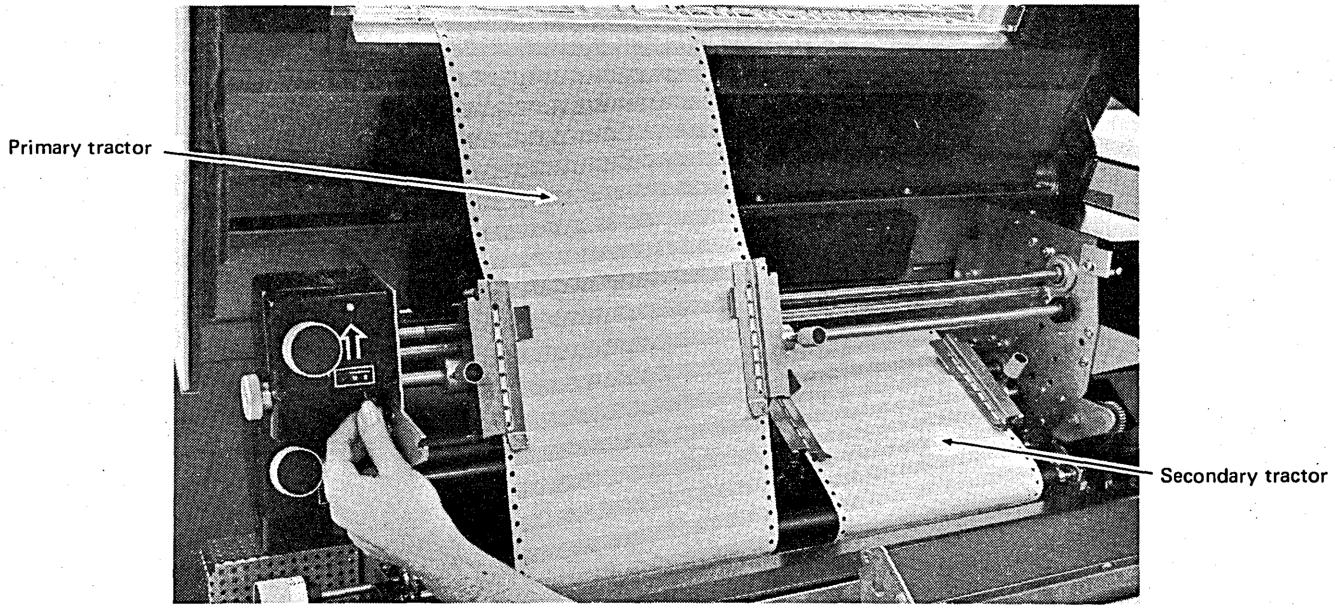


Figure 3-33. 2222 Printer with Dual Feed Tractors

54019A

Model 10 Card System, Model 10 Disk System, and Model 12

The feature of the 5203 printer that allows you to produce two separate printer output files is known as the Dual Feed Carriage Feature. The feature is available on 96, 120, and 132 position printers. One form is controlled by the left carriage of the printer and the other form is controlled by the right carriage. There is space between the right and left carriage tractors that contains no form. When you are printing on two forms you lose at least 17 print positions because no forms can be placed in this space (see *Output-Format Specifications,* later in this section).

Model 6

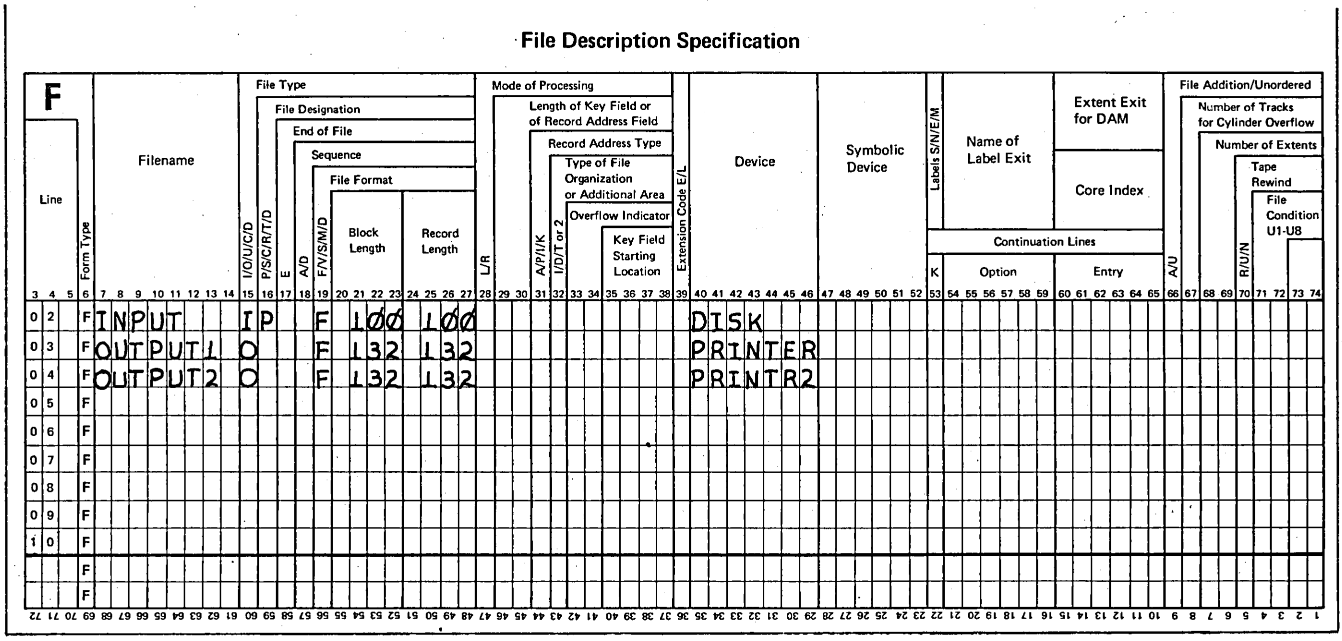
The 2222 printer on the Model 6 has dual tractor feeding, a *primary tractor* and a *secondary tractor.* All 220 positions of the printer are available for printing, that is, there need be no positions lost between the primary tractor and the secondary tractor.

The tractors that control the right-hand and left-hand forms on the printer are adjustable. That is, the width of the forms and the starting and ending print positions for each form are variable.

To print two output files for one program, each of the two printer files is considered a separate output file and must be described as such. These output files require special descriptions on the File Description and Output-Format sheets.

File Description Specifications

Model 10 Card System, Model 10 Disk System, and Model 12: Figure 3-34 is a sample File Description sheet for the Dual Feed Carriage Feature. The two output files to be printed must be assigned to the device names PRINTER and PRINTR2 on the File Description sheet (columns 40-46). PRINTER is the device name for the left carriage of the printer. The right carriage is assigned the device name PRINTR2. Record Length entries (columns 24-27) for the two printer files should be the same. Entries under Block Length (columns 20-23) must be the same as Record Length entries. You are responsible for ensuring that output to the PRINTER device is confined to the left-hand form and output to the PRINTR2 device is confined to the right-hand form. You can easily lay out your two reports using the Printer Spacing Chart.



File Description Specification

Line (3-5)	Form Type (6)	Filename (7-14)	File Type I/O/U/C/D (15)	File Designation P/S/C/R/T/D (16)	End of File E (17)	Sequence A/D (18)	File Format F/V/S/M/D (19)	Block Length (20-23)	Record Length (24-27)	Device (40-46)
02	F	INPUT	I	P			F	1ØØ	1ØØ	DISK
03	F	OUTPUT1	O				F	132	132	PRINTER
04	F	OUTPUT2	O				F	132	132	PRINTR2
05	F									
06	F									
07	F									
08	F									
09	F									
10	F									
	F									
	F									

F
Mode of Processing
Length of Key Field or of Record Address Field
Record Address Type
Type of File Organization or Additional Area
Overflow Indicator
Key Field Starting Location
Extension Code E/L
Symbolic Device
Labels S/N/E/M
Name of Label Exit
Extent Exit for DAM
Core Index
Continuation Lines
K
Option
Entry
A/U
File Addition/Unordered
Number of Tracks for Cylinder Overflow
Number of Extents
Tape Rewind
R/U/N
File Condition U1-U8
L/R
A/P/I/K
I/D/T or 2

Figure 3-34. File Descriptions for Two Printer Files on Model 10 Card System, Model 10 Disk System and Model 12 (5203 Printer)

Model 6: Figure 3-35 shows the File Description sheet entries for the dual tractors on the 2222 printer. The two printer files are assigned device names TRACTR1 and TRACTR2 on the File Description sheet (columns 40-46). TRACTR1 is the primary tractor (left-hand print unit) and TRACTR2 is the secondary tractor (right-hand print unit). Under Block Length (columns 20-23) for each file enter the *beginning print position* for that file. Under Record Length (columns 24-27) enter the *ending print position* for that file. Print positions for TRACTR1 and TRACTR2 cannot overlap.

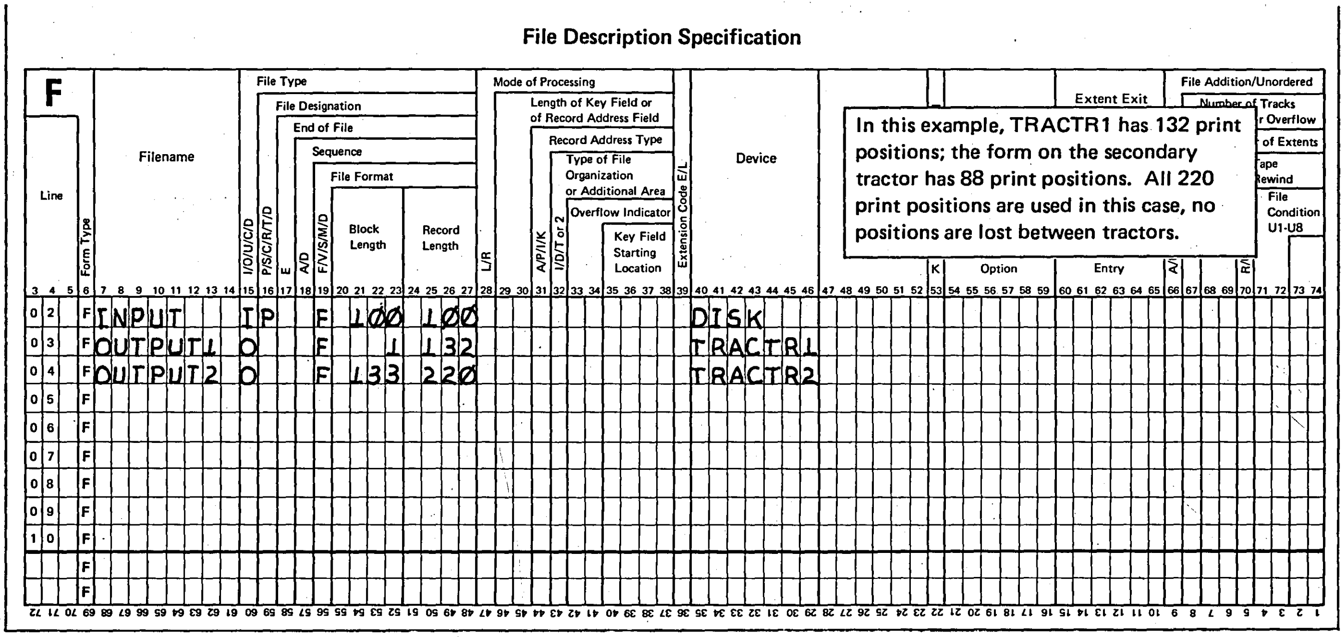
Output-Format Specifications

Spacing and skipping on the two forms are completely independent. You can specify different spacing and skipping for each output file. Spacing and skipping are entered in columns 17-22 of the Output-Format sheet.

Model 10 Card System, Model 10 Disk System and Model 12: Remember, there are 17 print positions you cannot use, because there is a space between the left and right carriage tractors which cannot contain a form. This is important when you are planning where to position your printing on each form. The first character to be printed on the form in the right carriage (PRINTR2) must be *at least* 17 positions away from the last character on the form in the left carriage (PRINTER). Suppose you decide to use print positions 1 through 80. Because the first character in the right carriage must be at least 17 positions away from the last character in the left carriage, print position 98 is the first available position:

```
  80  (End position of the form in the left carriage)
 +17  (Number of print positions you cannot use)
  97
```

Therefore, 98 is the first available position. If the length of your print line is 35 characters, the last position for the second form is 132.

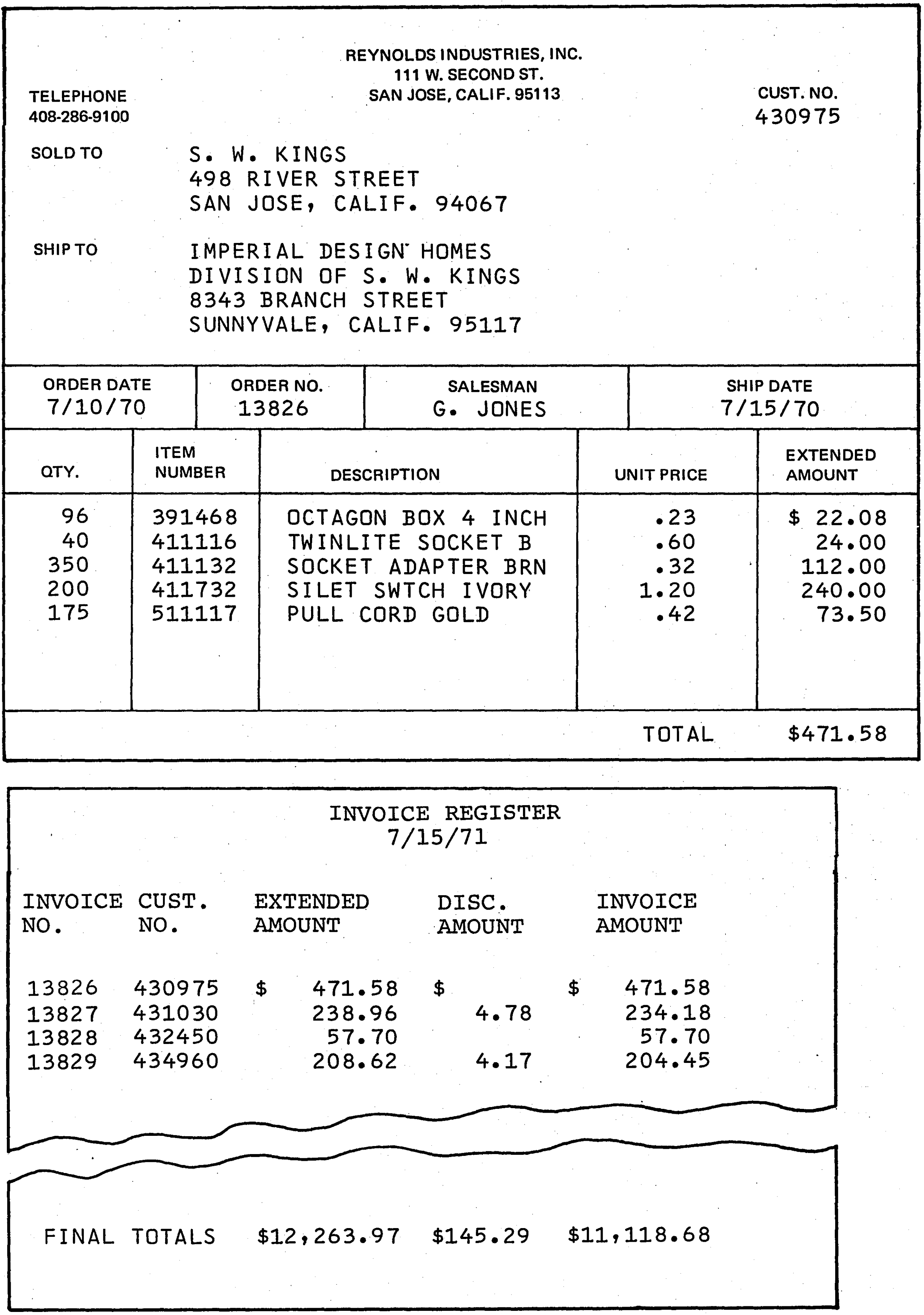


File Description Specification

In this example, TRACTR1 has 132 print positions; the form on the secondary tractor has 88 print positions. All 220 print positions are used in this case, no positions are lost between tractors.

Line	Form Type	Filename	File Type	File Designation	File Format	Block Length	Record Length	Device
02	F	INPUT	I	P	F	1ØØ	1ØØ	DISK
03	F	OUTPUT1	O		F	1	132	TRACTR1
04	F	OUTPUT2	O		F	133	22Ø	TRACTR2
05	F							
06	F							
07	F							
08	F							
09	F							
10	F							
	F							
	F							

Figure 3-35. File Descriptions for Two Printer Files on Model 6 (2222 Printer)



REYNOLDS INDUSTRIES, INC.
111 W. SECOND ST.
SAN JOSE, CALIF. 95113

TELEPHONE
408-286-9100

CUST. NO.
430975

SOLD TO
S. W. KINGS
498 RIVER STREET
SAN JOSE, CALIF. 94067

SHIP TO
IMPERIAL DESIGN HOMES
DIVISION OF S. W. KINGS
8343 BRANCH STREET
SUNNYVALE, CALIF. 95117

ORDER DATE	ORDER NO.	SALESMAN	SHIP DATE
7/10/70	13826	G. JONES	7/15/70

QTY.	ITEM NUMBER	DESCRIPTION	UNIT PRICE	EXTENDED AMOUNT
96	391468	OCTAGON BOX 4 INCH	.23	$ 22.08
40	411116	TWINLITE SOCKET B	.60	24.00
350	411132	SOCKET ADAPTER BRN	.32	112.00
200	411732	SILET SWTCH IVORY	1.20	240.00
175	511117	PULL CORD GOLD	.42	73.50
			TOTAL	$471.58

```
                    INVOICE REGISTER
                        7/15/71

INVOICE CUST.    EXTENDED     DISC.       INVOICE
NO.     NO.      AMOUNT       AMOUNT      AMOUNT

 13826  430975   $    471.58  $           $    471.58
 13827  431030        238.96       4.78        234.18
 13828  432450         57.70                    57.70
 13829  434960        208.62       4.17        204.45


  FINAL TOTALS   $12,263.97   $145.29     $11,118.68
```

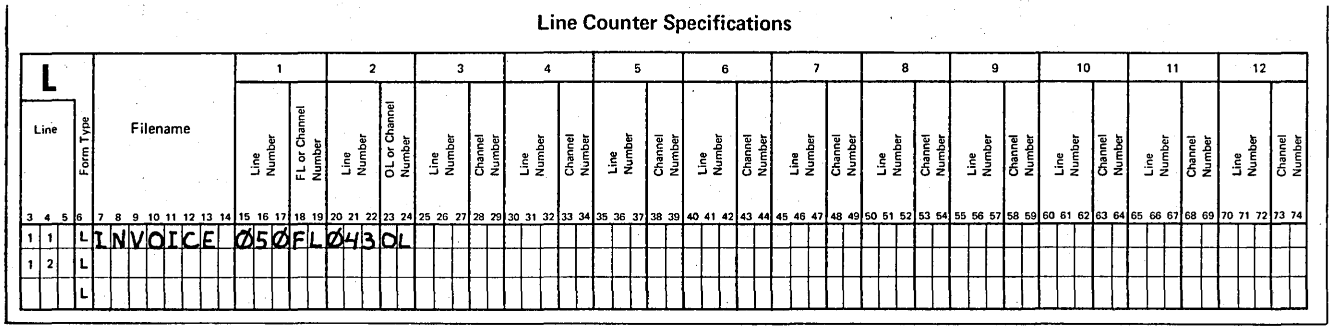
Figure 3-36. Sample Invoice and Invoice Register

Example: End-of-the-Month Billing

Assume that your company invoices its customers using your data processing system. It is your responsibility to prepare and print the invoices to be sent to the customers. You are also going to keep an invoice register; a record of every invoice that is sent out. Since you have a dual feed printer, you will print both the invoice and the invoice register at the same time. You might name your two output files INVOICE and INVREG.

The format of your output must be determined. In this case, INVREG will have the standard length of 66 lines, while INVOICE will have a nonstandard form length of 50. Heading information is printed on the top of each report. INVOICE uses a preprinted form for much of its heading information. INVOICE has a 63 print position line and INVREG has a 50 print position line. Figure 3-36, insert A is a sample invoice and Figure 3-36, insert B is a sample invoice register.

Since INVOICE has a nonstandard form length, it must be defined on the Line Counter sheet. You will use line 43 as the overflow line. Figure 3-37 shows a sample Line Counter sheet. (Note that since INVREG has a standard form length, it does not have to be defined on the Line Counter sheet.)



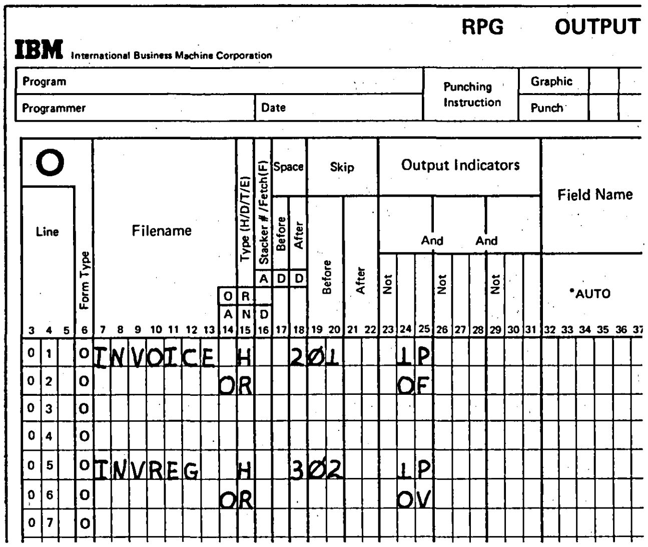
Line Counter Specifications

L			1		2		3		4		5		6		7		8		9		10		11		12	
Line	Form Type	Filename	Line Number	FL or Channel Number	Line Number	OL or Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
3 4 5	6	7 8 9 10 11 12 13 14	15 16 17	18 19	20 21 22	23 24	25 26 27	28 29	30 31 32	33 34	35 36 37	38 39	40 41 42	43 44	45 46 47	48 49	50 51 52	53 54	55 56 57	58 59	60 61 62	63 64	65 66 67	68 69	70 71 72	73 74
1 1	L	INVOICE	Ø5Ø	FL	Ø43	OL																				
1 2	L																									
	L																									

Figure 3-37. Line Counter Specifications for INVOICE

You also want to have headings printed on the top of every page. Because you do these operations only when an overflow indicator is on, you have to condition these operations by the overflow indicator. Figure 3-38 shows the File Description sheets for Model 10 (Insert A) and Model 6 (Insert B). Figure 3-39 shows the Output-Format sheet to print headings at the top of every page. (Remember that skipping and spacing on the two carriages are independent.)

| *Note for Model 10 and Model 12 Users:* Recall that the right form (in this case INVREG) must be at least 17 positions away from the form in the left carriage. Because INVOICE will have a 63 print position line, you assign positions 1-63 to it. INVREG is a 50 print position line and you assign it to positions 81 through 130.

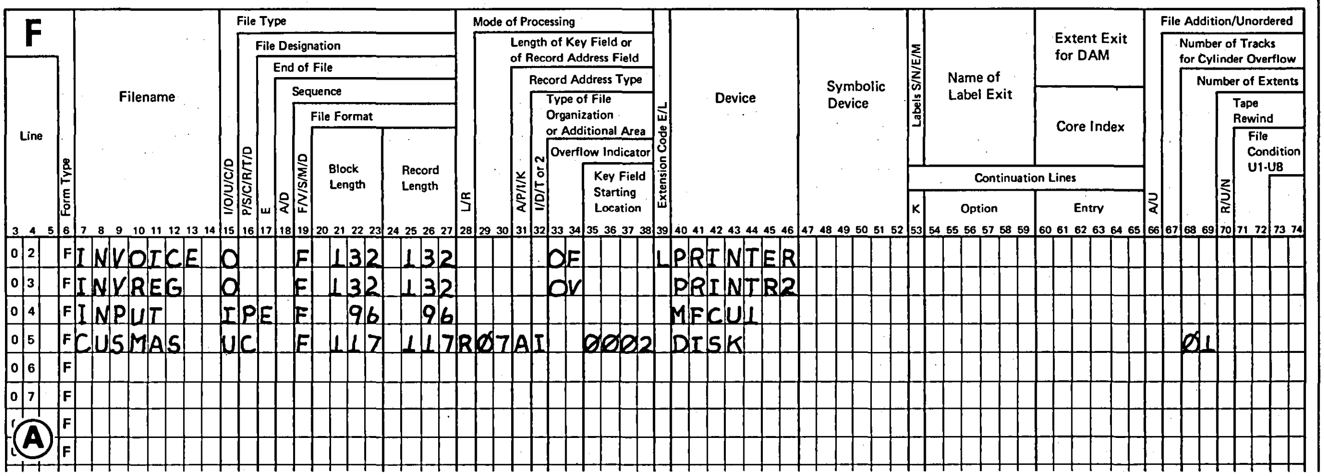


RPG OUTPUT

Line	Form Type	Filename	Type (H/D/T/E) / OR	Space After (18)	Skip Before (19-20)	Output Indicators (24-25)
01	O	INVOICE	H	2	Ø1	1P
02	O		OR			OF
03	O					
04	O					
05	O	INVREG	H	3	Ø2	1P
06	O		OR			OV
07	O					

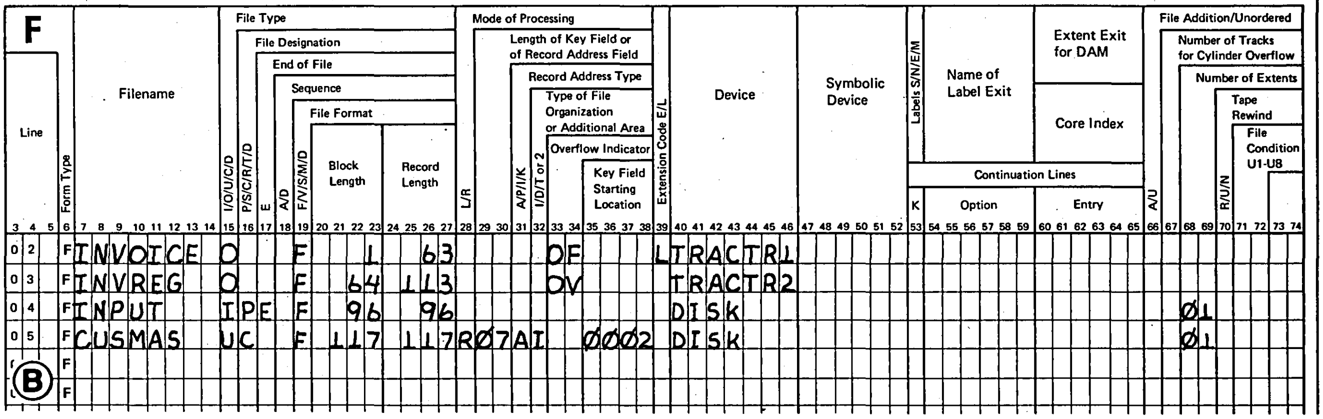
Figure 3-39. Heading Specifications for INVOICE and INVREG

File Description Specification



Line	Form Type	Filename	File Type (15)	File Designation (16)	End of File (17)	File Format (19)	Block Length	Record Length	L/R (28)	Length of Key Field (29-30)	Record Address Type (31)	Type of File Organization (32)	Overflow Indicator (33-34)	Key Field Starting Location	Extension Code (39)	Device	Number of Extents (68-69)
02	F	INVOICE	O			F	132	132					OF		L	PRINTER	
03	F	INVREG	O			F	132	132					OV			PRINTR2	
04	F	INPUT	I	P	E	F	96	96								MFCU1	
05	F	CUSMAS	U	C		F	117	117	R	Ø7	A	I		ØØØ2		DISK	Ø1
06	F																
07	F																
Ⓐ	F																
	F																

File Description Specification



Line	Form Type	Filename	File Type (15)	File Designation (16)	End of File (17)	File Format (19)	Block Length	Record Length	L/R (28)	Length of Key Field (29-30)	Record Address Type (31)	Type of File Organization (32)	Overflow Indicator (33-34)	Key Field Starting Location	Extension Code (39)	Device	Number of Extents (68-69)
02	F	INVOICE	O			F	1	63					OF		L	TRACTR1	
03	F	INVREG	O			F	64	113					OV			TRACTR2	
04	F	INPUT	I	P	E	F	96	96								DISK	Ø1
05	F	CUSMAS	U	C		F	117	117	R	Ø7	A	I		ØØØ2		DISK	Ø1
Ⓑ	F																
	F																

Figure 3-38. File Description Sheet for End-of-Month Billing

Overflow and Fetch Overflow

1. When you are not using overflow indicators or line counter specifications, but are allowing RPG II to handle overflow automatically, how many lines are assumed per page? What is the first line printed? What is the overflow line?

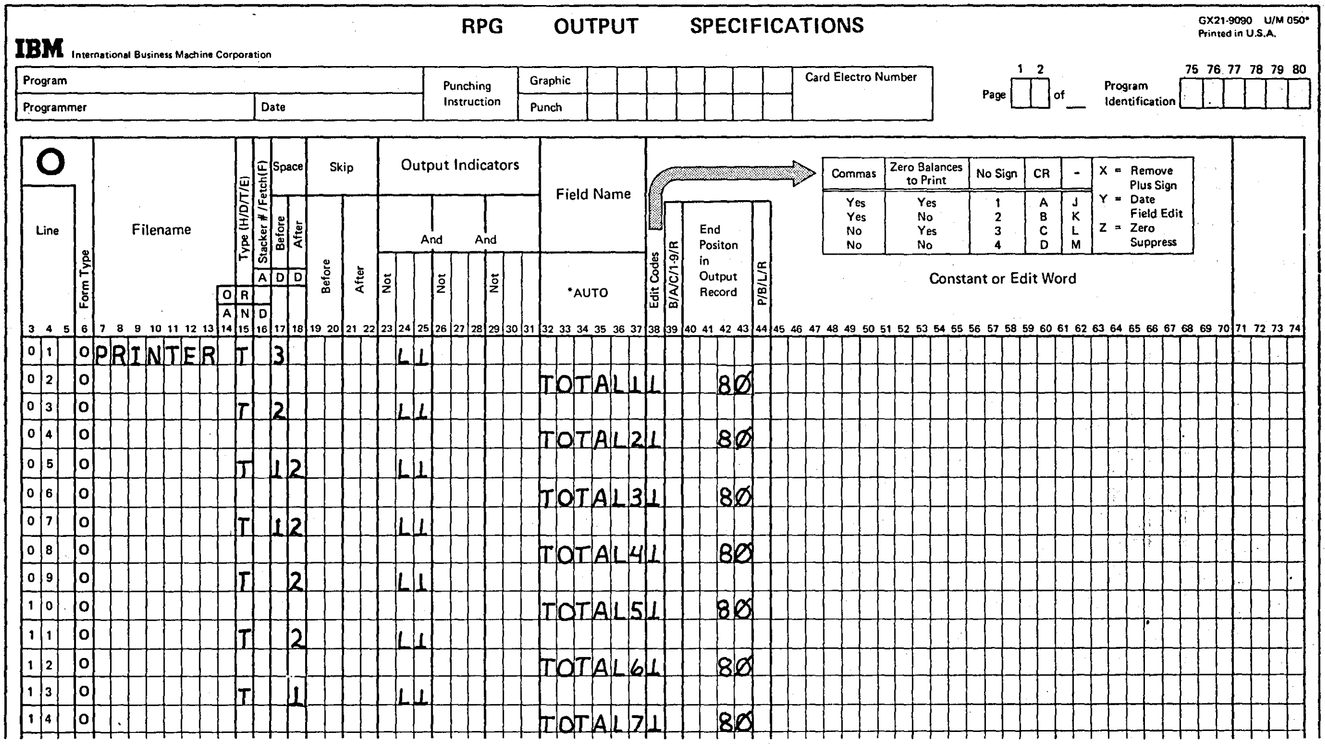
2. Code the line counter specifications which are necessary to define a form of 50 lines with the overflow line 8 lines from the bottom. What entries must also be made on the File Description sheet?

3. When is the overflow indicator turned on and when is it tested?

4. Describe a situation where printing can occur below the overflow line.

5. How does the fetch overflow specification alter the normal program cycle?

6. Given the following information, supply the Fetch specifications, for the output shown in Figure 3-40, which will prevent printing records on or over the perforation.
   - Number or printing lines per page is 66. The overflow line is 58.
   - There are seven total lines in all. Since all are conditioned by the same control level indicator, they will all print when a level 1 control break occurs.
   - Overflow should be forced if the overflow line is printed *prior* to beginning total output.
   - Total lines 1, 2, and 3, must print on the same page. Total lines 4 through 7 must print on the same page.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program

Programmer

Date

Punching Instruction

Graphic

Punch

Card Electro Number

Page of

Program Identification

O

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Output Indicators	Field Name	Edit Codes	End Position in Output Record
01	O	PRINTER	T	3		L1			
02	O						TOTAL1	1	80
03	O		T	2		L1			
04	O						TOTAL2	1	80
05	O		T	1	2	L1			
06	O						TOTAL3	1	80
07	O		T	1	2	L1			
08	O						TOTAL4	1	80
09	O		T		2	L1			
10	O						TOTAL5	1	80
11	O		T		2	L1			
12	O						TOTAL6	1	80
13	O		T		1	L1			
14	O						TOTAL7	1	80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Constant or Edit Word

- **Figure 3-40. Total Specifications (Question 6)**

Forms Alignment

7. Why is accurate forms alignmentt important?

8. What entries must be made in the RPG II specification sheets to repeat printing the first heading line of a report until the forms have been correctly aligned?

Editing

9. Choose the correct edit codes for the following punctuation.
   a. 1,342,650.00CR (for zero balance, print .00)
   b. 1,246,900– (for zero balance, leave the field blank)
   c. 1694824.25– (for zero balance, leave the field blank)
   d. 12/13/71

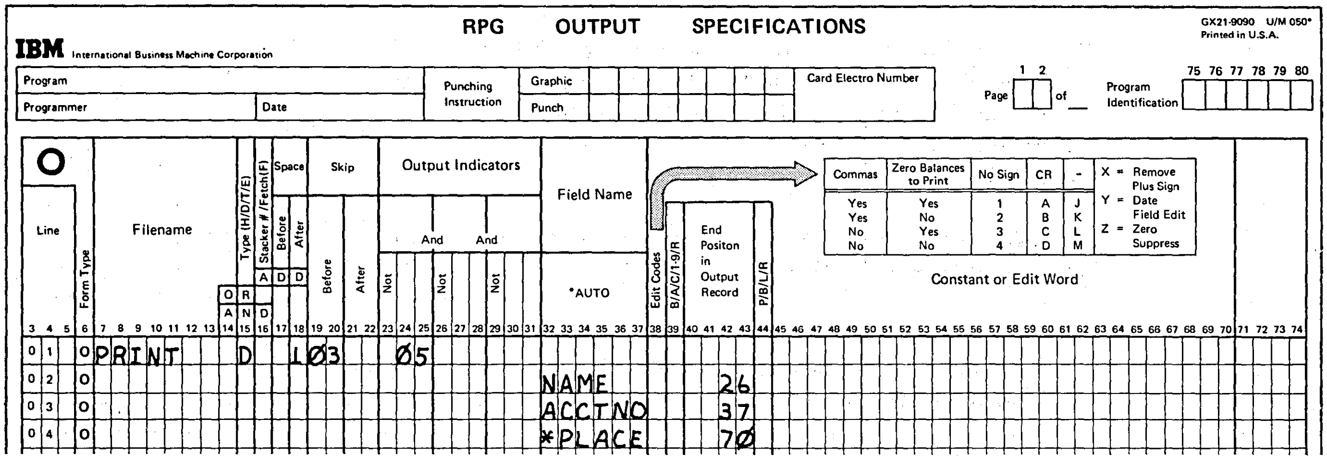
10. Construct an edit word for a 12-digit account number so that it will print out with the format XXX XXX XXX XX-X.

11. On the Output-Format sheet, specify the edit code and any other entries necessary to print out a dollar amount with the format $**1,234.56CR. The field must end in position 50. If the tield is zero, print 00.

*PLACE

12. What is the function of *PLACE?

13. In the example shown, is *PLACE used correctly? If not, why not?



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	End Position in Output Record
01	O	PRINT	D	1	Ø3	Ø5		
02	O						NAME	26
03	O						ACCTNO	37
04	O						*PLACE	7Ø

14. Write the output specifications to print two mailing labels in a row using *PLACE. The first label ends in print position 25, the second in 60. Each mailing label will have three lines and look like this:

NAME	(25 characters)		
ADDR1	(25)		
ADDR2	(18)	ZIPCODE	(5)

Dual Printer Output Files

15. What does the use of the dual feed printer allow you to do?

16. What limitations exist when designing forms for use with the Dual Feed Carriage Feature on the 5203 printer?

17. How do you distinguish between the two feeds on the RPG II specification sheets?

Answers To Review 3

1. Sixty-six lines are assumed per page. First line printed is 06 and the overflow line is 60.

2.

File Description Specification

F Line	Form Type	Filename	File Type I/O/U/C/D	File Designation P/S/C/R/T/D	End of File E	Sequence A/D	File Format F/V/S/M/D	Block Length	Record Length	L/R	Length of Key Field or of Record Address Field	Record Address Type A/P/I/K	Type of File Organization or Additional Area I/D/T or 2	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind R/U/N	File Condition U1-U8
3 4 5	6	7 8 9 10 11 12 13 14	15	16	17	18	19	20 21 22 23	24 25 26 27	28	29 30	31	32	33 34	35 36 37 38	39	40 41 42 43 44 45 46	47 48 49 50 51 52	53	54 55 56 57 58 59	60 61 62 63 64 65	66		67 68	69	70	71 72
02	F	PRINT	O						96					OF		L	PRINTER										

Line Counter Specifications

L Line	Form Type	Filename	1 Line Number	1 FL or Channel Number	2 Line Number	2 OL or Channel Number	3 Line Number	3 Channel Number	4 Line Number	4 Channel Number	5 Line Number	5 Channel Number	6 Line Number	6 Channel Number	7 Line Number	7 Channel Number	8 Line Number	8 Channel Number	9 Line Number	9 Channel Number	10 Line Number	10 Channel Number	11 Line Number	11 Channel Number	12 Line Number	12 Channel Number
3 4 5	6	7 8 9 10 11 12 13 14	15 16 17	18 19	20 21 22	23 24	25 26 27	28 29	30 31 32	33 34	35 36 37	38 39	40 41 42	43 44	45 46 47	48 49	50 51 52	53 54	55 56 57	58 59	60 61 62	63 64	65 66 67	68 69	70 71 72	73 74
1 1	L	PRINT	Ø5Ø	FL	Ø42	OL																				
1 2	L																									
	L																									

Form length is 50 lines and overflow line is 42. Any overflow indicator OA-OG or OV must be entered in columns 33-34 of the File Description sheet. L must be entered in column 39 to indicate that Line Counter specifications are used.

3. If the printer has printed on the overflow line either during total or detail output, the overflow indicator specified in the File Description sheet is turned on. The overflow indicator is tested and overflow output performed only after total output.

4. a. When more than one detail or total line is printed during one cycle and a line other than the last total line is printed on the overflow line.
   b. When the last detail line for a control group prints on the overflow line, the total lines for that group will print below the overflow line.

5. The overflow indicator is tested prior to printing each line specified with fetch overflow rather than waiting until all total output has occurred. If the indicator is on when tested, overflow output is performed immediately and then the line specified is printed.

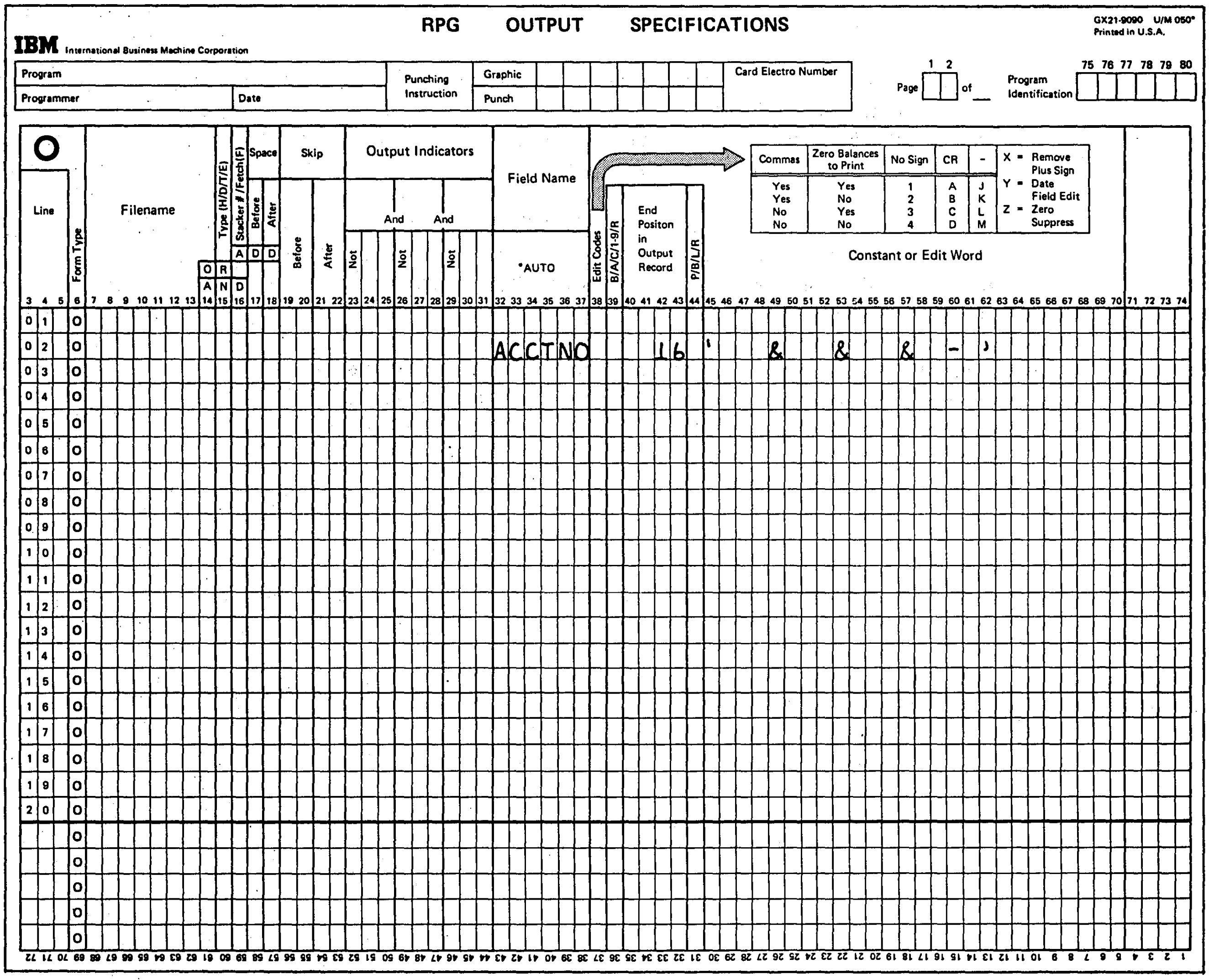
6. F in column 16 of lines 1 and 7 of Figure 3-40. F in column 16 of line 1 causes forms to advance if the overflow indicator is on, assuring that total 1 will not print below the overflow line and total 3 will not fall over the perforation. Since totals 4 through 7 must all print on the same page and will not all fit below the overflow line, enter F in the specifications for total 4 to cause a skip to the next page if the overflow indicator is on. Since totals 5-7 must print on the same page as total 4, no fetch specification should be entered for them.

7. Forms must be aligned accurately so that printing is correctly positioned on each page and so that printing occurs exactly where you want it, not above or below.

8. There must be at least one line on the output sheets conditioned by the 1P indicator. A '1' must be entered in column 41 of the control card.

9. a. A
   b. K
   c. M
   d. Y

10. Blank spaces show where digits are to be placed and &'s show where blanks go. The entire word must be enclosed in quotes.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

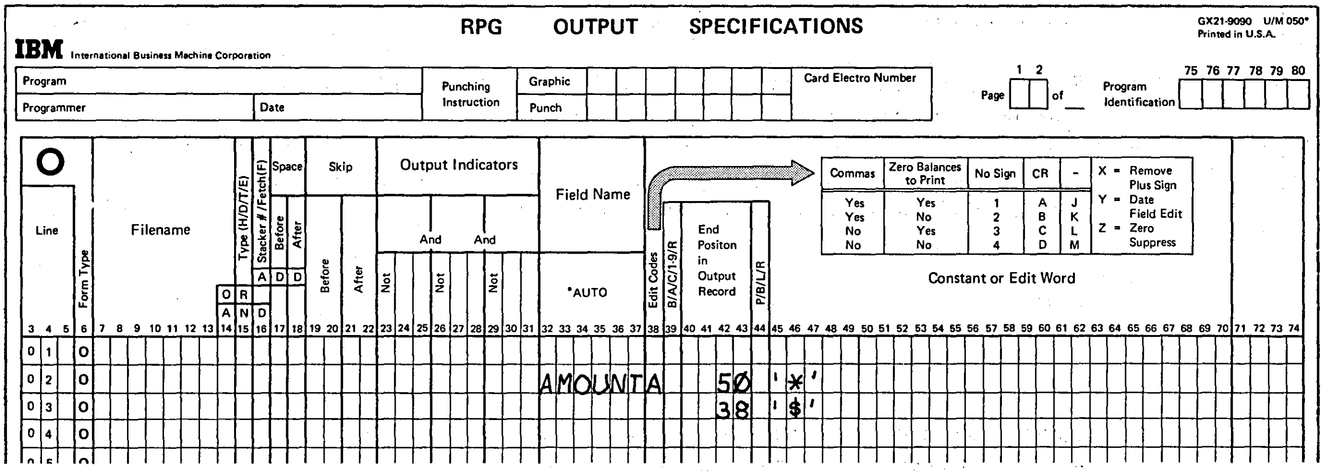
Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 □□ of __
Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Field Name (32-37)	End Position in Output Record (40-43)	Constant or Edit Word (45-70)
01	O			
02	O	ACCTNO	16	' & & & - '
03	O			
04	O			
05	O			
06	O			
07	O			
08	O			
09	O			
10	O			
11	O			
12	O			
13	O			
14	O			
15	O			
16	O			
17	O			
18	O			
19	O			
20	O			
	O			
	O			
	O			
	O			
	O			

11. The A edit code will zero suppress and insert commas, decimal points, and the credit sign CR. The asterisk entered in columns 45-47 will cause all places zero suppressed to be filled with asterisks. The dollar sign must be specified on the line following the edit code. When printed in position 38, it will come right before the asterisks.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification
Programmer | Date

O

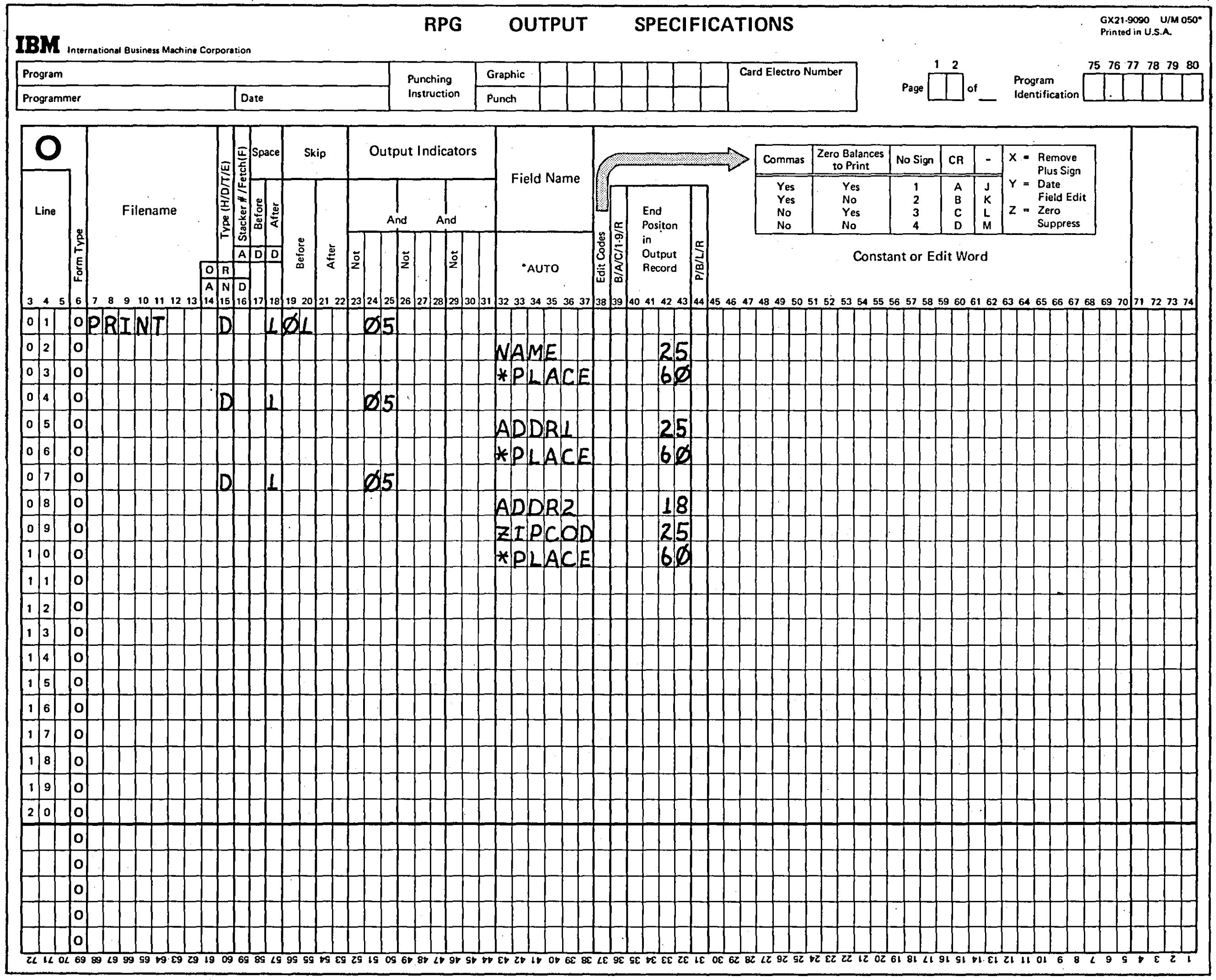
Line	Form Type	Filename	Type (H/D/T/E)	Stacker # /Fetch(F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name *AUTO	Edit Codes	B/A/C/1-9/R	End Positon in Output Record	P/B/L/R	Constant or Edit Word
01	O														
02	O									AMOUNT	A		50		'*'
03	O												38		'$'
04	O														

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

12. The function of *PLACE is to easily code the printing of duplicate information on the same output line. *PLACE places information from print position 1, through the highest end position previously defined for a field into the print positions indicated by the end position in the *PLACE entry.

13. It is not correct. The end positon in the *PLACE specification is not high enough. The duplicated information will overlay the field called ACCTNO. The end position on the *PLACE line should be at least twice the highest end position previously specified for that record.

14. Two labels must be printed. Therefore, for each line you must specify the original field and a *PLACE entry, which will cause the contents of the original field to be duplicated.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program ______ Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of ___ | Program Identification 75 76 77 78 79 80

Programmer ______ Date ______

Commas / Zero Balances to Print / No Sign / CR / - : Yes Yes 1 A J; Yes No 2 B K; No Yes 3 C L; No No 4 D M. X = Remove Plus Sign, Y = Date Field Edit, Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	B/A/C/1-9/R (39)	End Position in Output Record (40-43)	P/B/L/R (44)	Constant or Edit Word (45-70)
01	O	PRINT	D		1	Ø1		Ø5						
02	O								NAME			25		
03	O								*PLACE			6Ø		
04	O		D		1			Ø5						
05	O								ADDR1			25		
06	O								*PLACE			6Ø		
07	O		D		1			Ø5						
08	O								ADDR2			18		
09	O								ZIPCOD			25		
10	O								*PLACE			6Ø		
11	O													
12	O													
13	O													
14	O													
15	O													
16	O													
17	O													
18	O													
19	O													
20	O													
	O													
	O													
	O													
	O													
	O													

15. The dual feed carriage allows the printing of two independent reports simultaneously.

16. A minimum of seventeen print positions must be left blank between the two output forms.

17. *Model 10 Card System, Model 10 Disk System, and Model 12:* The only difference is that the device name on the File Description sheet for the left carriage is PRINTER and for the right carriage is PRINTR2.

Model 6: The only difference is that the device name on the File Description sheet for the left tractor is TRACTR1 and for the right tractor is TRACTR2.

Chapter 4. Card Output Operations

CHAPTER 4 DESCRIBES:

Puncned output.

Printing on cards.

Using one file for both input and output.

Selecting the stacker for output cards.

Merging input and output file cards.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Using the printer to produce a simple listing.

Using control fields.

RPG II object program cycle for detail and total operations.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Coding for punching a combined or output card file; summary punching.

Formatted printing and unformatted printing (*PRINT) on cards.

Uses and coding for combined files.

Stacker selecting input, combined, and output card files.

Coding for merging input and output card files.

Note: You can use the review questions contained in *Review 4* at the end of this chapter to test your comprehension of each topic in the chapter. Questions are grouped according to the topic to which they apply. Answers follow the review questions.

INTRODUCTION

Thus far, in this manual, the program output usually has been a printed form or report. Punched cards generally have been used as a source of input data. However, cards can be used for output as well as input. This chapter describes RPG II coding for output operations using the IBM 5424 MFCU, available on the System/3 Models 10 and 15, and the IBM 2560 MFCM, available on the System/3 Model 15 only.

You might want to have card output for many reasons. Perhaps you want to generate a new file or change an input file in some way, such as by reformatting the records, adding data, deleting data, adding new records, or deleting unwanted records. The output you choose might be data punched on the cards, printed on the cards, or both. Information can be printed on cards for identification, interpretation of the punched data, or any other purpose you desire. You can punch and print data on blank cards or on cards that already contain data. You can also direct cards to a specific stacker or more output file cards with cards from an input file.

PUNCHING AND PRINTING ON CARDS

Punching and printing on individual output cards are controlled separately. Punched cards need not be printed; printed cards need not be punched.

Punched Output

Punched output can be used to:

- Create a file of cards that is different from the input card file
- Add new records to a card file
- Add fields to input records
- Punch a summary card from a group of input cards

RPG II coding for punched output is similar to coding for printer output. Either the primary hopper (device name MFCU1 or MFCM1) or the secondary hopper (device name MFCU2 or MFCM2) can be used as the output device; the other hopper is used as the input device. In some cases, punching can be done on the input cards themselves (see *Using One File for Both Input and Output,* in this chapter). Remember, however, if only one MFCU hopper is used, it must be MFCU1 (Model 10 Card System only).

On the Output-Format sheet, you can specify heading, detail, and total output, just as you can with a printer file. In some cases, you may want to punch only total records by summing the data on several input records and punching a separate card for the total. This is known as *summary punching.* Do not specify an edit code for punched output unless you want to have punctuation punched into the output cards.

Printing On Cards

It is advantageous to print the same information on the card as was punched on the card. This way you can easily interpret what information is recorded on the card. Also, printing information on the card makes it easier to recreate a card that is damaged to the extent that it cannot be read by the MFCU or MFCM, or duplicated by the data recorder.

Although you may print the same information on the card that is punched on the card, it is not always necessary to do so. You may print entirely different fields from those that are punched.

The 96-column card has space at the top for four lines of printing (Figure 4-1). Each line can contain 32 printed characters, for a total of 128 print positions.

The 80-column card can be printed only on the MFCM Model A1 with the optional print feature. Up to six print lines can be used. Each line can contain up to 64 printed characters, for a total of 384 print positions.

The MFCM print heads can be set to print in 25 different print line positions, from above the 12-punch position to below the 9-punch positions (Figure 4-2). The print heads, numbered 1 through 6, must remain in sequence from top to bottom, with print head 1 at the top. Therefore, with six print heads installed, print head 1 cannot be set below line 20 and print head 6 cannot be set above line 6. Intermediate line positions are located on and between each row of punch positions. Print-position 5 (between the 11-row and 0-row) should be avoided, if possible, because the feed wheel may cause some smudging of characters printed in that position. In punched fields, printing in even-numbered line positions should be avoided because punching may obliterate some characters.

Formatted Printing (MFCU)

Using formatted printing, you may print a field or constant in any of the 128 print positions available on a 96-column card. The first three lines are the lines usually printed. The fourth is printed only if necessary because printing on the fourth line increases considerably the amount of time needed to print, and thus increases the time needed to do the job.

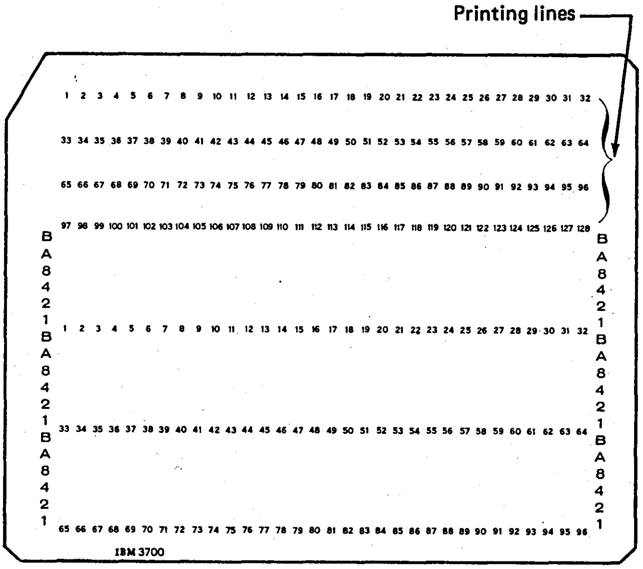
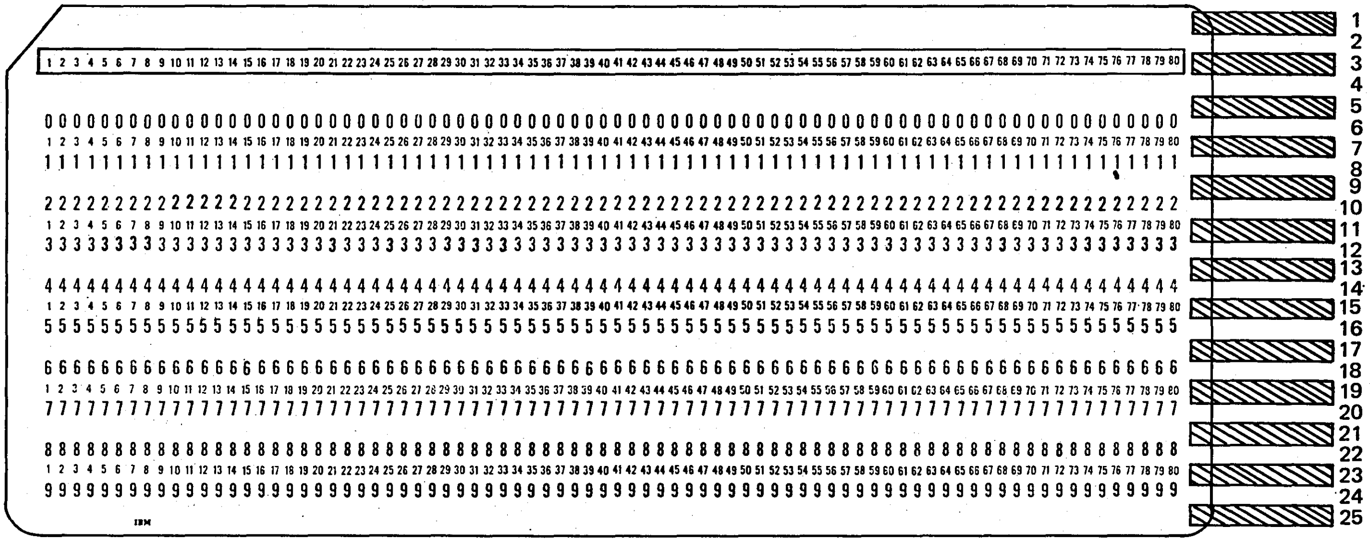


Figure 4-1. Printing Lines on a 96-Column Punch Card



● **Figure 4-2. Printing Lines on an 80-Column Punch Card**

For each field or constant you wish printed by the MFCU, you must make the following specifications on the Output-Format sheet:

1. If a field is to be printed, enter the field name in columns 32-37.

2. Enter * in column 40 to indicate that the field is to be printed not punched.

3. Enter any end position from 001 to 128 in columns 41-43.

4. If a constant is to be printed, enter the constant in columns 45-70.

For example, to print the fields on the card shown in Figure 4-3, the specifications in Figure 4-4 lines 06-10 are necessary. You indicate punching of these fields by specifying an end position without the asterisk (see Figure 4-4, lines 02-05). If you intend to punch fields and print fields, you need two specifications per field. If you have seven fields to be both punched and printed, you need 14 specifications.

Because printing and punching are two separate functions, it is possible to print a field without punching it and punch a field without printing it. If you punch and print the same field, you may put each in different positions. In other words, you may format the punched fields in a different way than you format the printed fields.

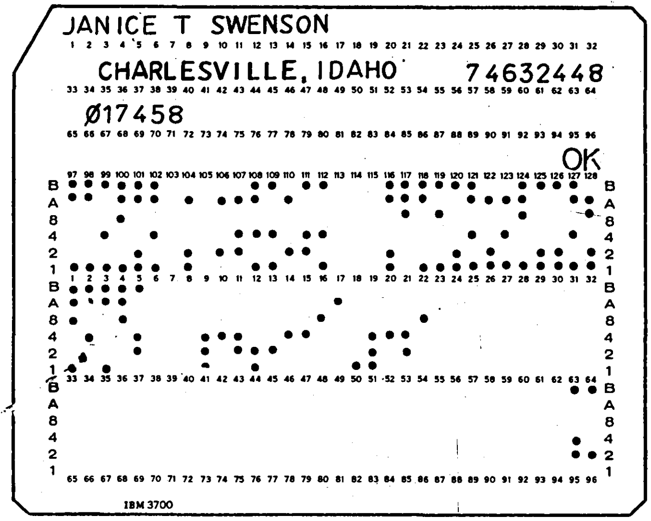
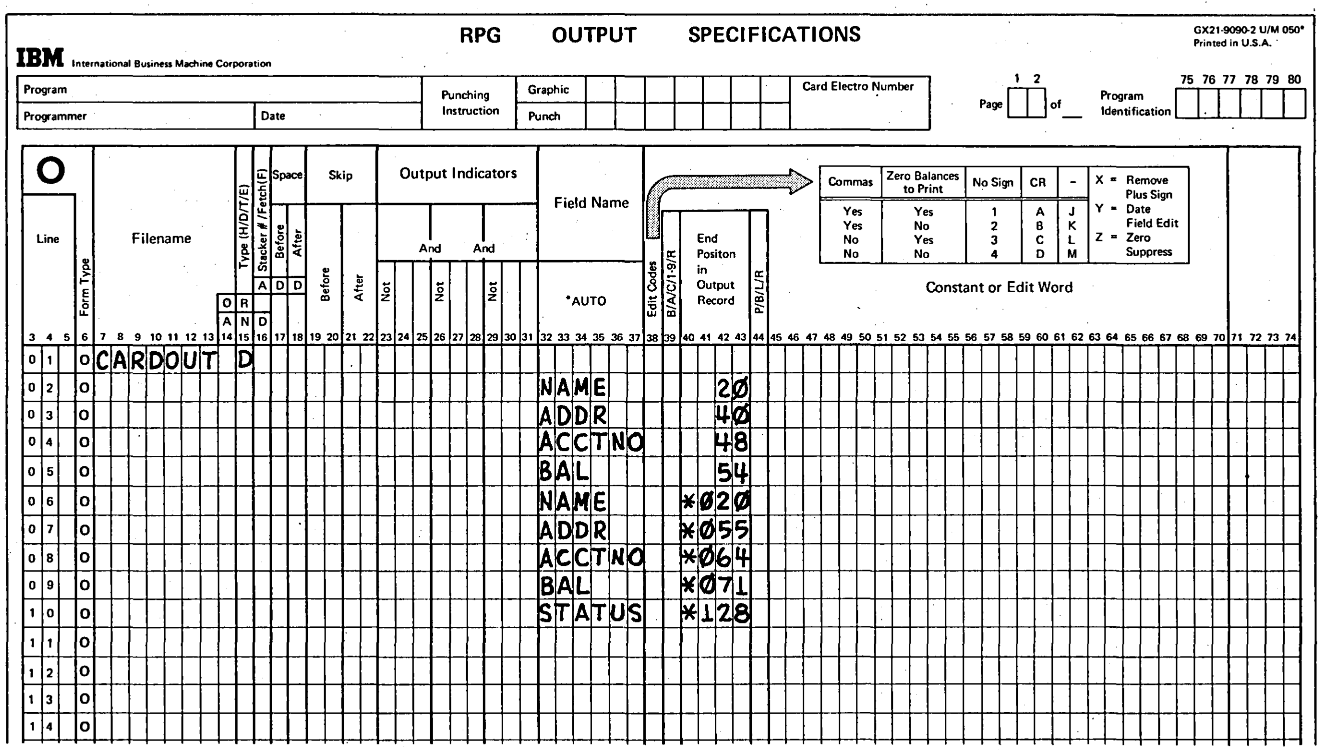


Figure 4-3. Formatted Printing on a 96-Column Card



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090-2 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page ☐☐ of
Program Identification

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Field Name (32-37)	B/A/C/1-9/R (39)	End Position in Output Record (40-43)	Constant or Edit Word (45-70)
01	O	CARDOUT	D				
02	O			NAME		2Ø	
03	O			ADDR		4Ø	
04	O			ACCTNO		48	
05	O			BAL		54	
06	O			NAME		*Ø2Ø	
07	O			ADDR		*Ø55	
08	O			ACCTNO		*Ø64	
09	O			BAL		*Ø71	
10	O			STATUS		*128	
11	O						
12	O						
13	O						
14	O						

Figure 4-4. Specifications for Punching and Printing on a Card (Formatted Printing — MFCU)

Formatted Printing – MFCM

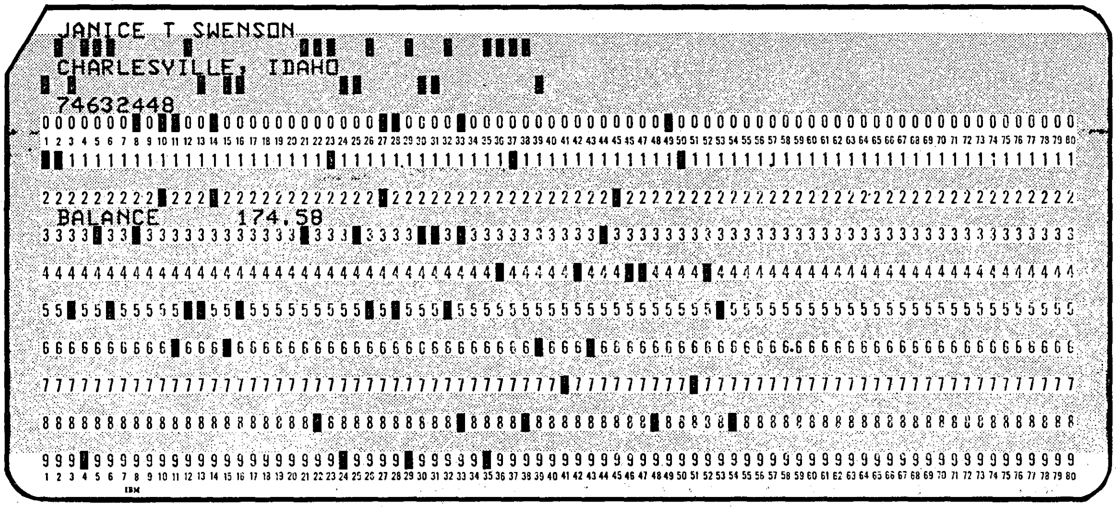
Using formatted printing, you can print a field or constant on any six of the 25 print lines available on an 80-column card. For each field or constant you wish printed by the MFCM, you must make the following specifications on the Output-Format sheet:

1. When printing a field, enter the field name in columns 32-37.
2. Specify a print head number (1-6) in column 41.
3. Specify a print end-position (01-64) in columns 42 and 43. (The leading zero is required when specifying print positions 01-09.)
4. When printing a constant, enter the constant in columns 45-70.

For example, to print the fields shown in Figure 4-5, the specifications shown in Figure 4-6 are necessary. Coding lines 02 through 05 cause the fields to be punched. Coding lines 06 through 10 cause the fields and a constant, BALANCE, to be printed. As you can see, two specifications are required for each field that is to be both punched and printed. In order to obtain the desired printing results, the MFCM print heads must be aligned mechanically prior to running the program.

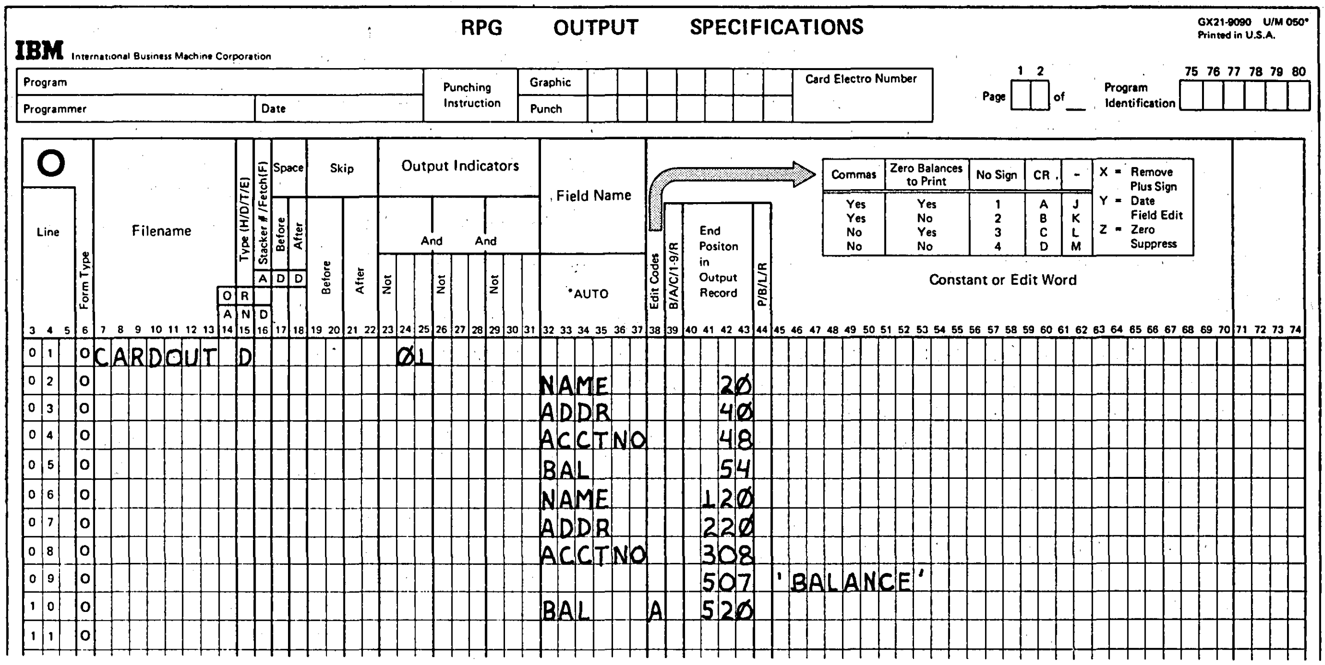
Note: The fourth line of printing also could have been printed using print head 4, with print heads 5 and 6 set to line positions lower on the card.

Because printing and punching are two separate functions, it is possible to print a field without punching it and to punch a field without printing it.



JANICE T SWENSON
CHARLESVILLE, IDAHO
74632448
BALANCE 174.58

● **Figure 4-5. Formatted Printing on an 80-Column Card**



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	Constant or Edit Word
01	O	CARDOUT	D	Ø1					
02	O				NAME			2Ø	
03	O				ADDR			4Ø	
04	O				ACCTNO			48	
05	O				BAL			54	
06	O				NAME			12Ø	
07	O				ADDR			22Ø	
08	O				ACCTNO			3Ø8	
09	O							5Ø7	'BALANCE'
10	O				BAL	A		52Ø	
11	O								

- Figure 4-6. Specifications for Punching and Printing on a Card (Formatted Printing — MFCM)

*Unformatted Printing (*PRINT) — MFCU and MFCM*

Using formatted printing, recall that if you wish to both punch and print a field, you must have two entries per field on the Output-Format sheet: a punch entry and a print entry. If you want several fields to be both punched and printed, there is a great deal of coding involved.

RPG II has a special reserved word, *PRINT, which allows you to punch and print fields and constants with less coding. When *PRINT is specified, it causes all previous fields described for the record to be printed. Use of *PRINT is known as unformatted printing.

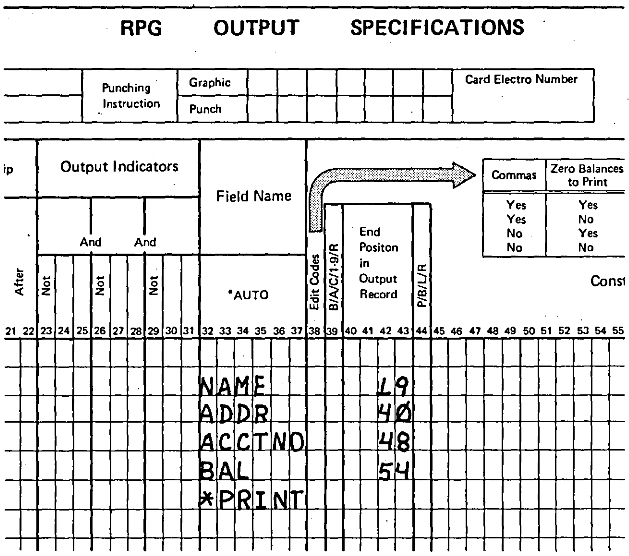
Figure 4-7 shows the use of the *PRINT specification. NAME, ADDR, ACCTNO, and BAL are to be punched. Following these field names in columns 32-37 is the entry *PRINT. This entry causes the previous four fields to also be printed on the card (see Figures 4-8 and 4-9).

Using *PRINT with the MFCU causes fields and constants to be printed in positions corresponding to the punch positions. For example, ACCTNO is punched in positions 41-48 and also is printed in positions 41-48.

On the MFCM, there is not space to print 80 characters on one line. Therefore, data punched in columns 1-64 is printed in print positions 1-64 by print head 1; data punched in columns 65-80 is printed in positions 49-64 by print head 2 (Figure 4-9).

The word *PRINT can be used only once for a record and must be entered after the description of all fields that are to be both punched and printed. Suppose, instead of printing all four fields (NAME, ADDR, ACCTNO, and BAL), you want to print only NAME and ACCTNO. In this case, the entry *PRINT must follow NAME and ACCTNO. ADDR and BAL must be described after the *PRINT entry (Figure 4-10).

Columns 7-22 and 39-74 must always be left blank on the *PRINT specification line.



RPG OUTPUT SPECIFICATIONS

Punching Instruction	Graphic	Punch	Card Electro Number

Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Positon in Output Record	P/B/L/R
	*AUTO				
	NAME			19	
	ADDR			4Ø	
	ACCTNO			48	
	BAL			54	
	*PRINT				

Commas	Zero Balances to Print
Yes	Yes
Yes	No
No	Yes
No	No

ip
After
Not
And
Not
And
Not
Const
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

Figure 4-7. Punching and Printing on a Card Using *PRINT (Unformatted Printing)

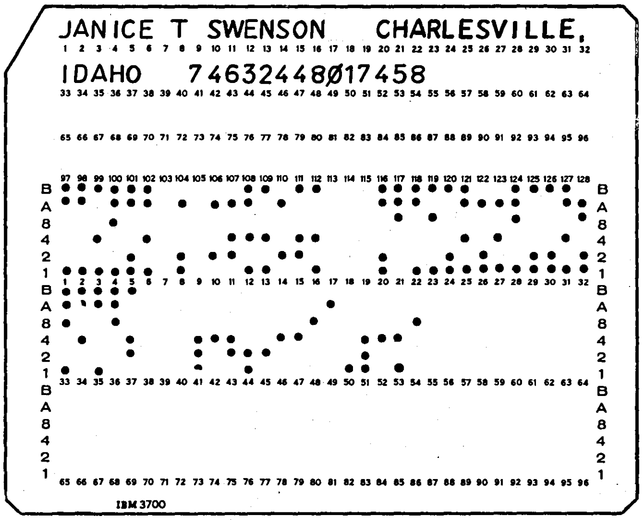


Figure 4-8. Unformatted Printing on a 96-Column Card

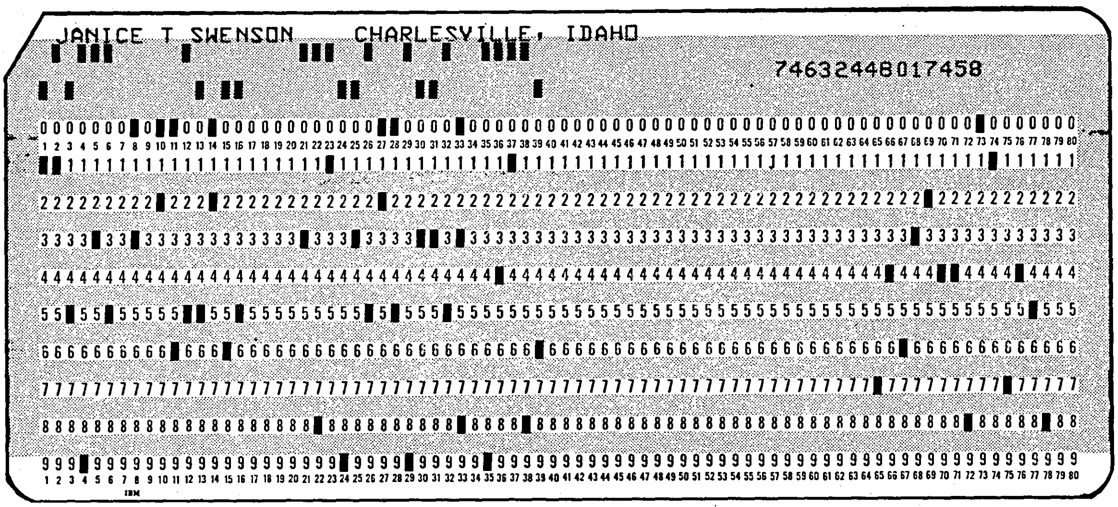
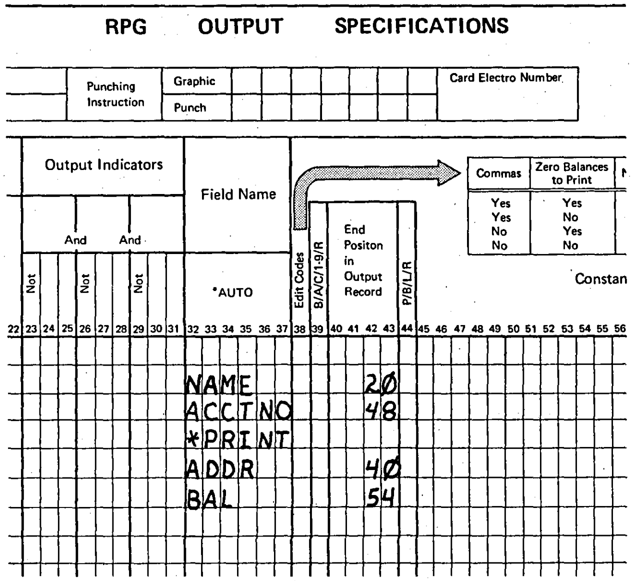
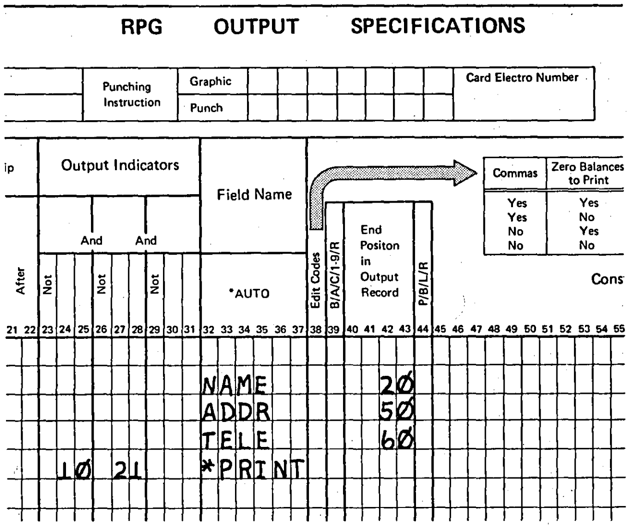


Figure 4-9. Unformatted Printing on an 80-Column Card


RPG OUTPUT SPECIFICATIONS
Punching Instruction
Graphic
Punch
Card Electro Number
Output Indicators
And
And
Not
Not
Not
Field Name
*AUTO
Edit Codes
B/A/C/1-9/R
End Positon in Output Record
P/B/L/R
Commas
Zero Balances to Print
Yes Yes
Yes No
No Yes
No No
Constan
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56

Field Name	End Position in Output Record
NAME	2Ø
ACCTNO	48
*PRINT	
ADDR	4Ø
BAL	54

Figure 4-10. Using *PRINT to Specify Fields When Some are Punched Only


RPG OUTPUT SPECIFICATIONS
Punching Instruction
Graphic
Punch
Card Electro Number
ip
Output Indicators
And
And
After
Not
Not
Not
Field Name
*AUTO
Edit Codes
B/A/C/1-9/R
End Positon in Output Record
P/B/L/R
Commas
Zero Balances to Print
Yes Yes
Yes No
No Yes
No No
Cons
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

Output Indicators		Field Name	End Position in Output Record
		NAME	2Ø
		ADDR	5Ø
		TELE	6Ø
1Ø	21	*PRINT	

Figure 4-11. Conditioning *PRINT Entry

You may use any indicator in columns 23-31 to condition the *PRINT entry. That specification will then be performed only when the condition set by the indicator is met. For example, according to the specification in Figure 4-11, line 05, the fields will be printed only when 10 and 21 are both on at the same time.

Editing A Field Printed on the Card

Any field that is to be printed, using either formatted or unformatted printing, can be edited. This, of course, will make the printed field easier to read and understand.

Editing a field to be printed on the card is done in the same way as editing a field to be printed on the printer. However, editing should be kept at a minimum so that the length of the printed field won't be considerably larger than the length of the punched field. Zero suppression or merely removal of the sign are often done since they do make the printing easier to read, but still keep the printing in a one-to-one relationship with the punches.

USING ONE FILE FOR BOTH INPUT AND OUTPUT

In your experience with RPG II, you have used card files as input files and output files. Suppose, however, you wanted to punch into the same card that was read into the computer. Is it possible to use input and output files to do this? No, input cards are only read and output cards are only punched or printed. What you need is a combination of the two. The RPG II language allows such a file combination. This type of file is called a *combined file.*

Note: In this discussion, only the MFCU is used for reference; unless noted otherwise, the MFCM can be used in the same way.

Punching Into the Same Card that is Read

A company keeps a daily record of the amount of each item sold. At the end of the week the daily amount sold for each item is punched into the card in six different fields, one field per day. These cards are then used in a program which totals the daily amount sold for each item and punches that total into the same card that contained the daily amounts.

Figure 4-12 shows the format of the data card which is read. Each card contains the end of the week date (DATE), the item number (ITEMNO) and daily amounts sold (FLD1 through FLD6). Figure 4-13 shows the same card after it has been punched. TOTAL is the field punched after the total amount sold has been calculated.

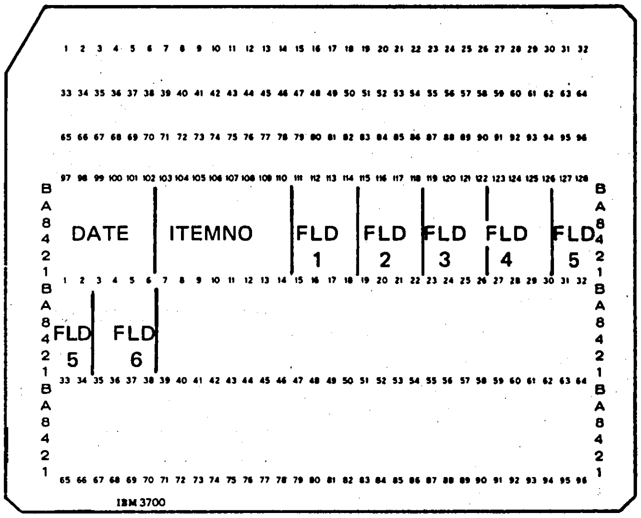


Figure 4-12. Combined File Card Read

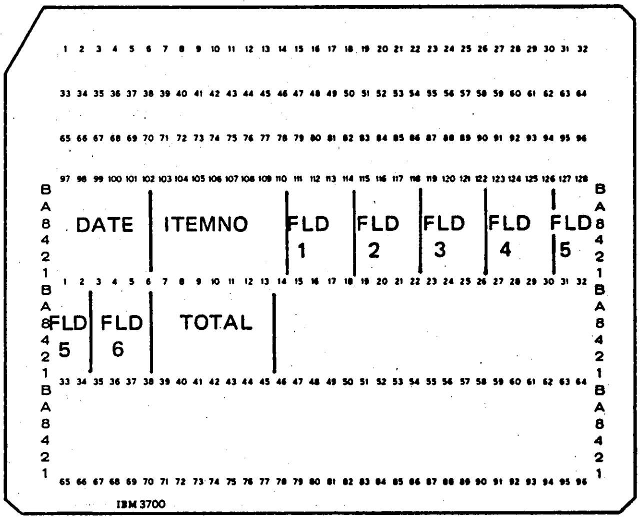


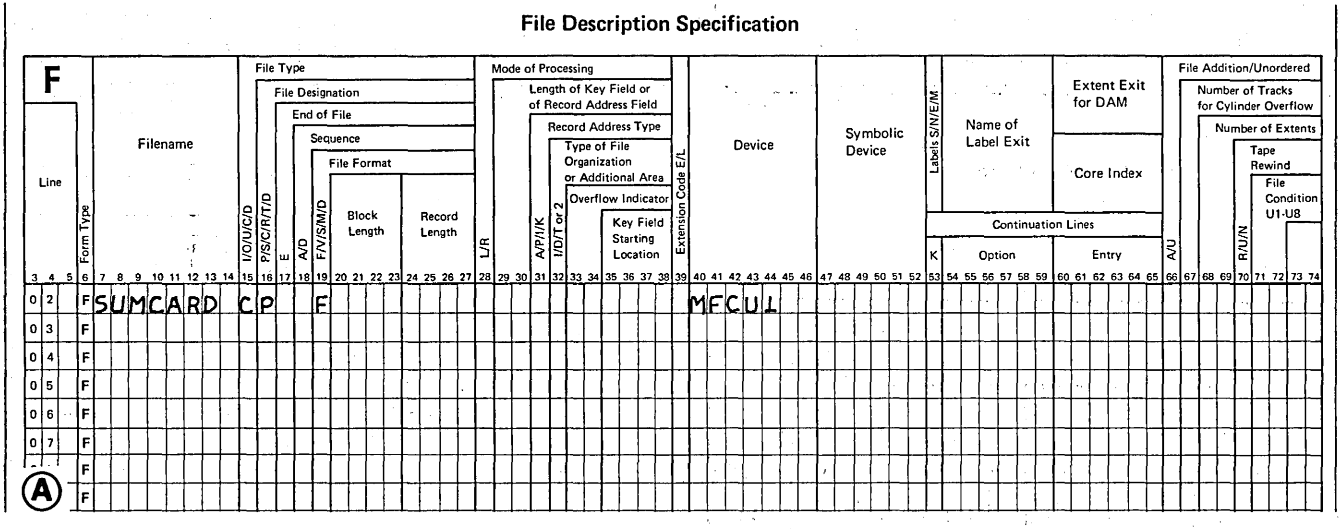
Figure 4-13. Combined File Card After Punching

To describe this program, you will need four types of specifications sheets: File Description, Input, Calculation, and Output-Format. On the File Description sheet, you define the file as a combined file with a letter C in column 15. File Description entires for a combined file are the same as those for an input file except for column 15. (See Figure 4-14, insert A.) Cards will be both read and punched on MFCU1. With the exception of the device name, specifications in Figure 4-11 apply to the MFCM, as well.

On the Input sheet (Figure 4-14, insert B) you should define only the fields which are to be read. Remember the TOTAL field is not in the cards when the cards are read and therefore, is not described on the Input sheet. Since the TOTAL field is to be created during the job, it is defined on the Calculation sheet (Figure 4-14, insert C).

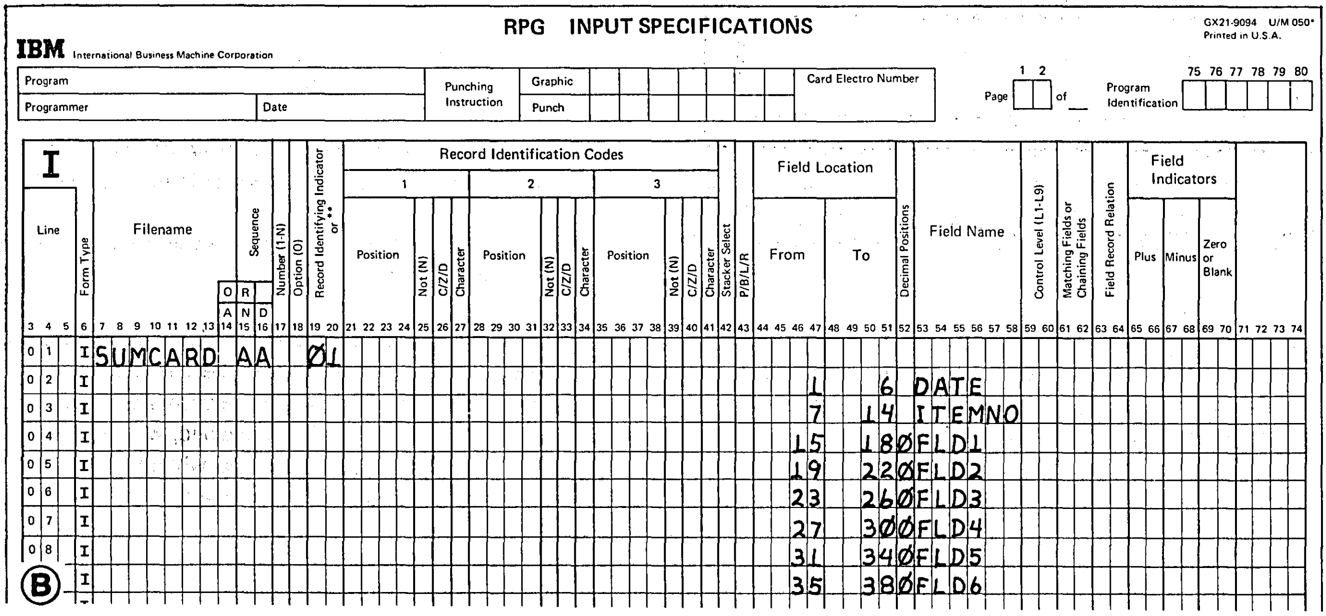
The Output-Format sheet (Figure 4-14, insert D) describes only the information that is to be punched into the card. Here again you describe the TOTAL field.

Note: Be sure the card columns to be punched contain blanks, to prevent problems with invalid punch combinations.



File Description Specification

Line	Form Type	Filename (7-14)	File Type (15)	File Designation (16)	File Format (19)	Device (40-46)
02	F	SUMCARD	C	P	F	MFCU1
03	F					
04	F					
05	F					
06	F					
07	F					
	F					
Ⓐ	F					



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program

Programmer Date

Punching Instruction Graphic Punch

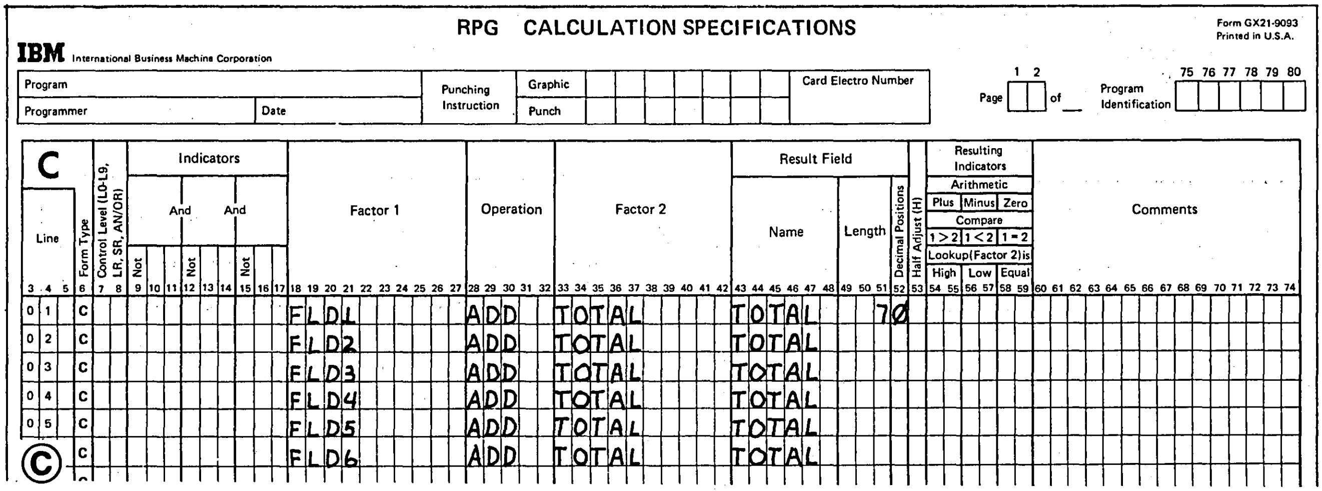
Card Electro Number

Page of

Program Identification

Line	Form Type	Filename (7-14)	Sequence (15-16)	Record Identifying Indicator (19-20)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)
01	I	SUMCARD	AA	Ø1				
02	I				1	6		DATE
03	I				7	14		ITEMNO
04	I				15	18	Ø	FLD1
05	I				19	22	Ø	FLD2
06	I				23	26	Ø	FLD3
07	I				27	30	Ø	FLD4
08	I				31	34	Ø	FLD5
Ⓑ	I				35	38	Ø	FLD6

Figure 4-14 (Part 1 of 2). Specifications for Reading and Punching Each Card

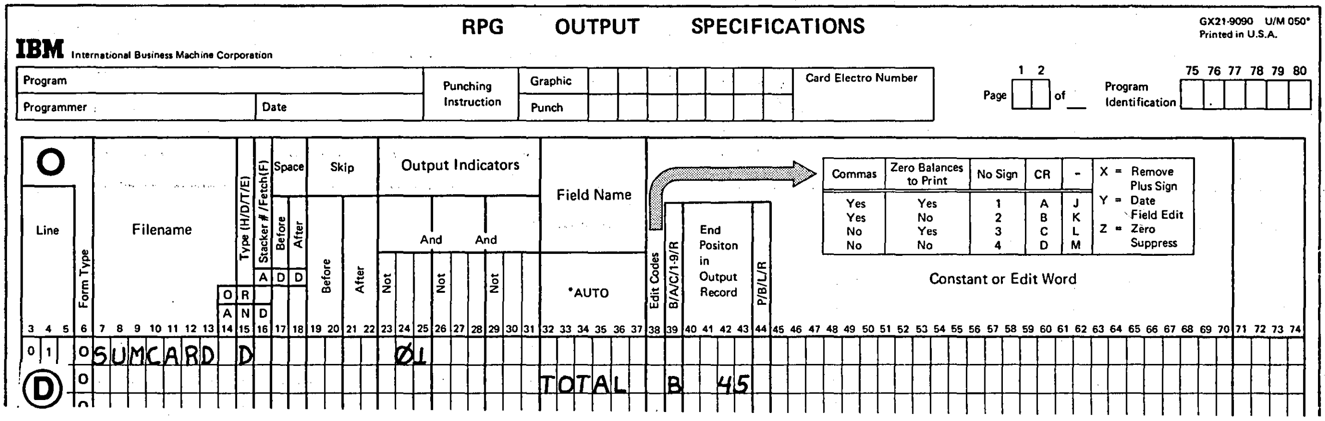


IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions
01	C	FLD1	ADD	TOTAL	TOTAL	7	Ø
02	C	FLD2	ADD	TOTAL	TOTAL		
03	C	FLD3	ADD	TOTAL	TOTAL		
04	C	FLD4	ADD	TOTAL	TOTAL		
05	C	FLD5	ADD	TOTAL	TOTAL		
Ⓒ	C	FLD6	ADD	TOTAL	TOTAL		



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	Blank After (B)	End Position in Output Record
01	O	SUMCARD	D	Ø1			
Ⓓ	O				TOTAL	B	45

Figure 4-14 (Part 2 of 2). Specifications for Reading and Punching Each Card

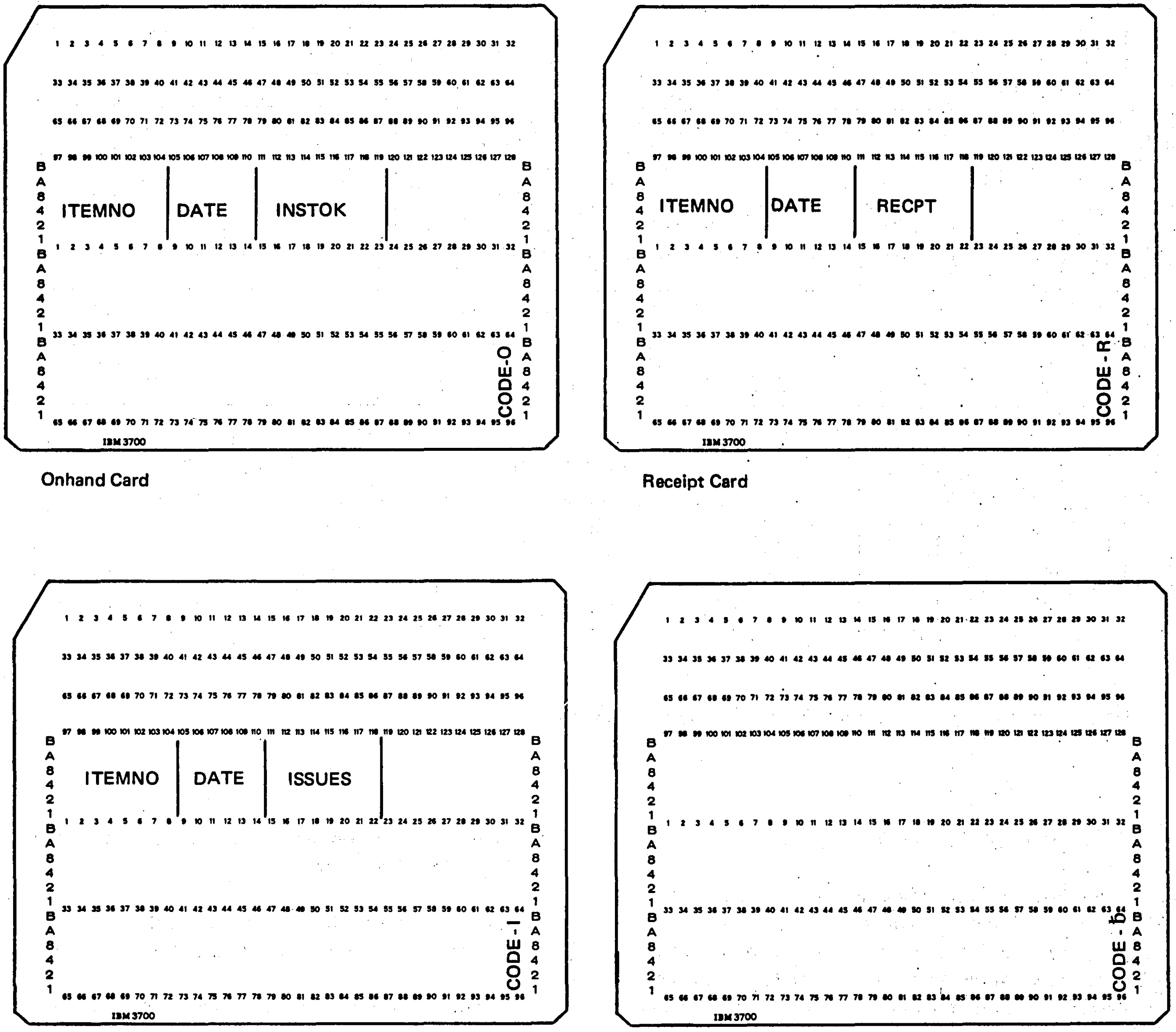
Punching into a Blank Card in the File

You have just learned how to use a combined file when you wish to read and punch the same card. What if you wish to read several cards and then punch another card in the same file? Remember any time you want to read and punch cards from the *same* file, that file must be defined as a combined file.

Assume that a company which keeps a weekly record of items sold, uses these records at the end of the month to determine the quantity of items on hand. For each item, four different types of cards are read (Figure 4-15).

1. Onhand, which contains the number in stock at the beginning of the month.
2. Issues, which contains the number sold during the month. There is one of these cards for each week.
3. Receipts, which contains the number added to stock through reorder.
4. New Onhand, which is blank. After calculations have been performed to determine the number on hand, this number and the date and a code are punched into the card. This card, when punched, will be used as next month's Onhand card, and will be in the same format as the current Onhand card.

Each of these card types is identified by a code in column 96 (see Figure 4-15). (This code would be column 80 if the MFCM were used.)



Onhand Card

Receipt Card

Issues Card

New Onhand Card

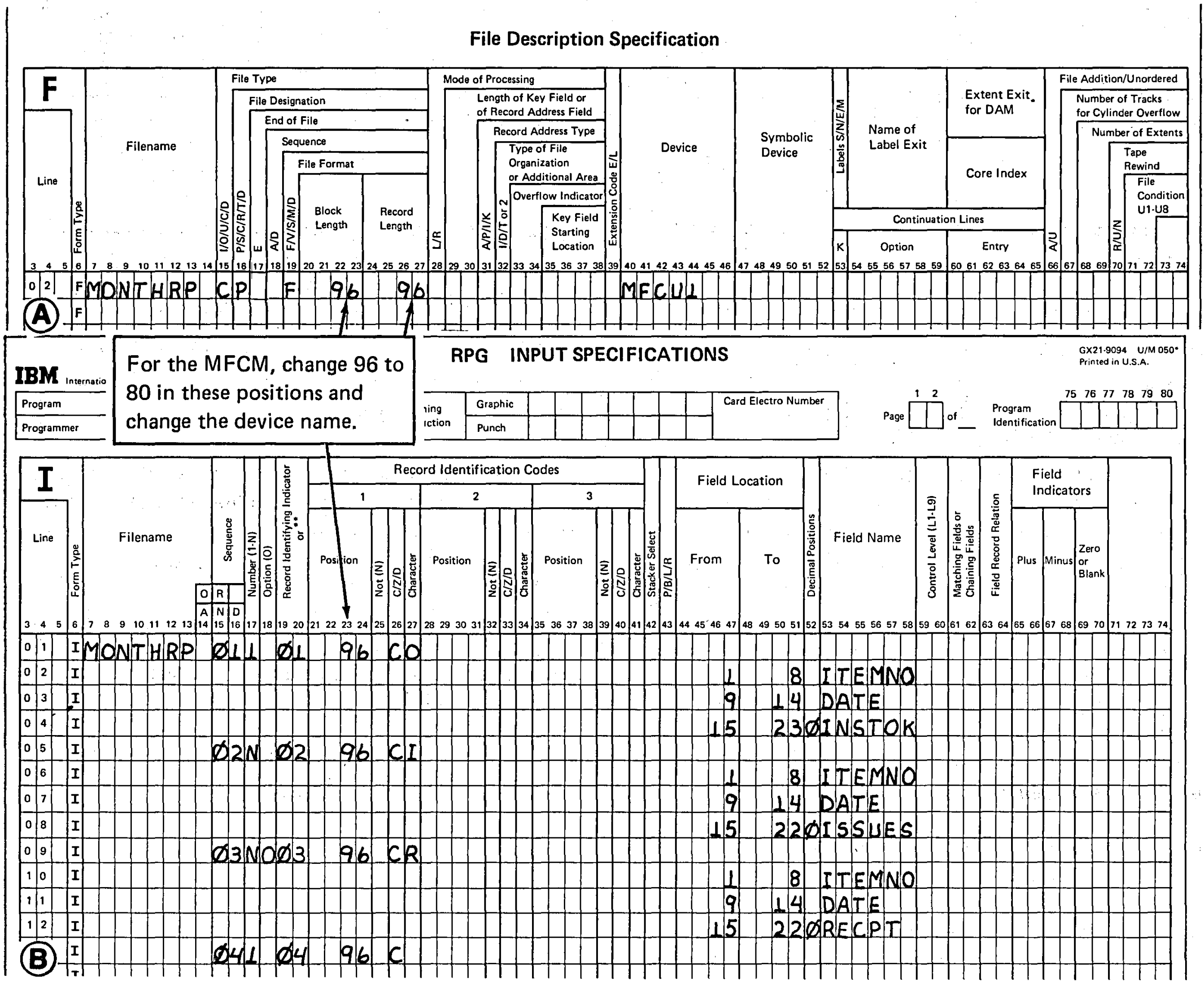
Figure 4-15. Four Card Types

Again, you will need four types of specification sheets to write your program: File Description, Input, Calculation, and Output-Format. On the File Description sheet, you must enter the filename, file type, and device (see Figure 4-16, insert A). Since this file is both read and punched, the file type must be C to denote a combined file.

On the Input sheet, you must describe all four card types and assign record identifying indicators. These indicators will be used later to condition those operations which are to occur only when a specific card type has been read.

These cards must be read in a certain order. The onhand card must come first, the new onhand (blank) card must come last. The issues and receipts cards may be in either order, but must always be in the same order for any program. For this program, assume that receipts cards follow issues. Remember to indicate that cards are to be in a certain sequence by using numeric sequence entries for all card types. These entries will direct the program to check for sequence. Figure 4-16, insert B, shows input specifications for this program.

On the Calculation sheet, you define the calculations (Figure 4-16, insert C) which must be performed to determine amount on hand. The record identifying indicators assigned on the Input sheet are used to condition calculations. For example, only when an issues card is read (02 is on) will number sold (ISSUES) be subtracted from INSTOK.



File Description Specification

Line	Form Type	Filename	File Type (I/O/U/C/D)	File Designation (P/S/C/R/T/D)	File Format (F/V/S/M/D)	Block Length	Record Length	Device
02	F	MONTHRP	C	P	F	96	96	MFCU1
	F							

A

For the MFCM, change 96 to 80 in these positions and change the device name.

IBM

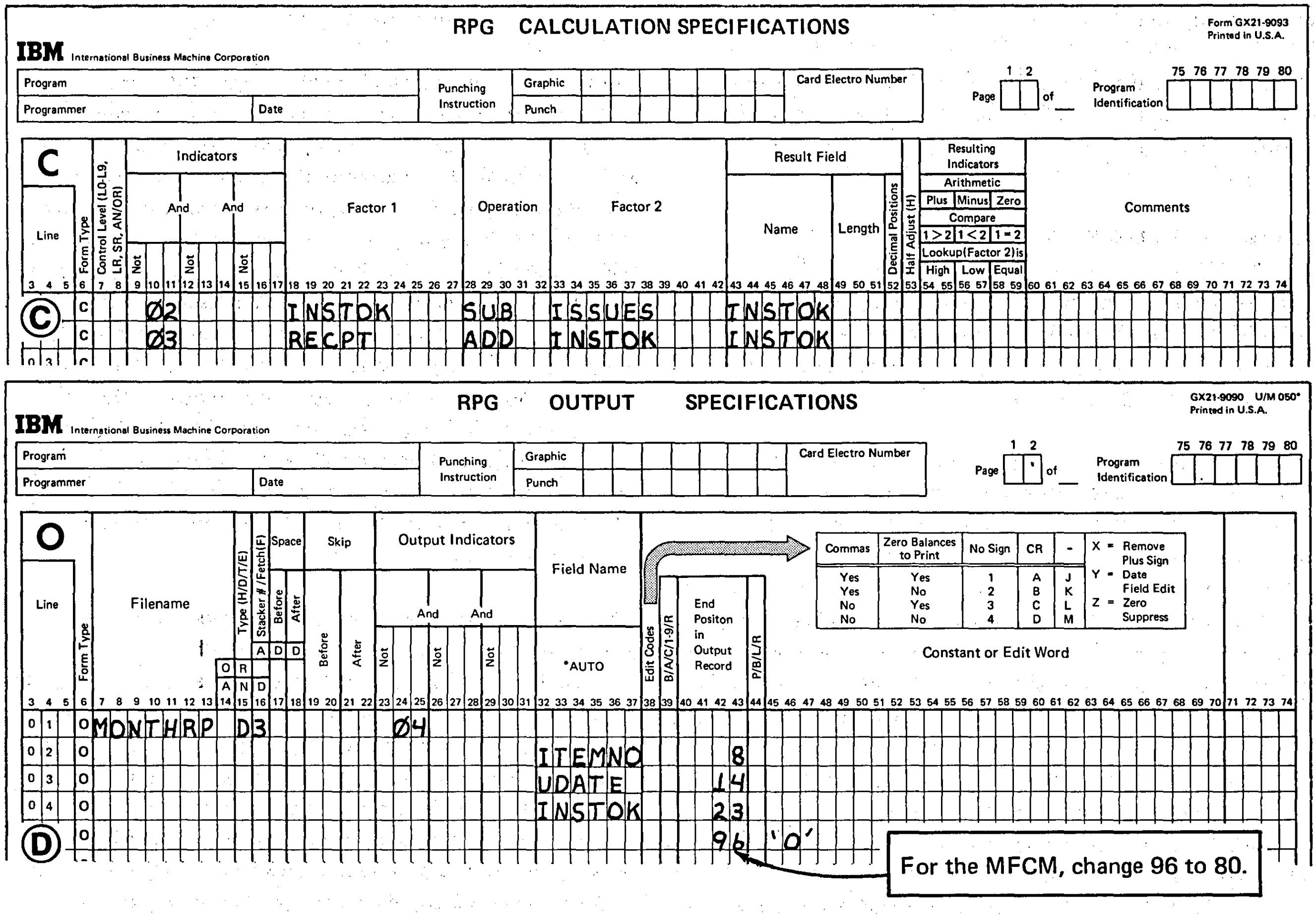
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	From	To	Decimal Positions	Field Name
01	I	MONTHRP	Ø1	1		Ø1	96		C	O				
02	I										1	8		ITEMNO
03	I										9	14		DATE
04	I										15	23	Ø	INSTOK
05	I		Ø2	N		Ø2	96		C	I				
06	I										1	8		ITEMNO
07	I										9	14		DATE
08	I										15	22	Ø	ISSUES
09	I		Ø3	N	O	Ø3	96		C	R				
10	I										1	8		ITEMNO
11	I										9	14		DATE
12	I										15	22	Ø	RECPT
	I		Ø4	1		Ø4	96		C					

B

Figure 4-16 (Part 1 of 2). Specifications for Reading and Punching Combined File Cards



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name
Ⓒ	C	Ø2	INSTOK	SUB	ISSUES	INSTOK
	C	Ø3	RECPT	ADD	INSTOK	INSTOK

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type	Stacker	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	MONTHRP	D	3	Ø4			
02	O					ITEMNO	8	
03	O					UDATE	14	
04	O					INSTOK	23	
Ⓓ	O						96	'0'

For the MFCM, change 96 to 80.

Figure 4-16 (Part 2 of 2). Specifications for Reading and Punching Combined File Cards

On the Output-Format sheet (Figure 4-16, insert D) you must show the fields which are to be punched. Since only the blank card is punched, punching occurs only when 04 is on. Because the card punched will be used as next month's Onhand card, ITEMNO, DATE, INSTOCK and a code must be punched.

When To Specify a Combined File

How do you know whether to describe a card file that must be read into the computer as an input or as a combined file? If you remember these basic rules you will have no trouble deciding.

1. If the file contains cards that are to be *both* read and punched, it must be a combined file.

2. If cards in the file are to be stacker selected on some basis other than card type, the file must be described as a combined file (see *Stacker Selection, Selecting on a Basis Other Than Card Type*).

STACKER SELECTION

Stacker selection is the means by which you can separate certain cards from all others in the file. If stacker selection entries are not made, all cards automatically fall into specific, predefined stackers:

MFCU1:	Stacker 1
MFCU2:	Stacker 4
MFCM1:	Stacker 1
MFCM2:	Stacker 4 (Model A2) or Stacker 5 (Model A1)

Note: Unless stated otherwise, this discussion applies to both the MFCU and the MFCM.

Input and Combined File Cards

Input file cards are stacker selected by input specifications. Combined file cards can be stacker selected by either input or output specifications.

Selecting on the Basis of Card Type

Stacker selection by input specifications must be on the basis of card type. This means you may separate all cards of one type from the input file by specifying a special stacker into which they should be stacked.

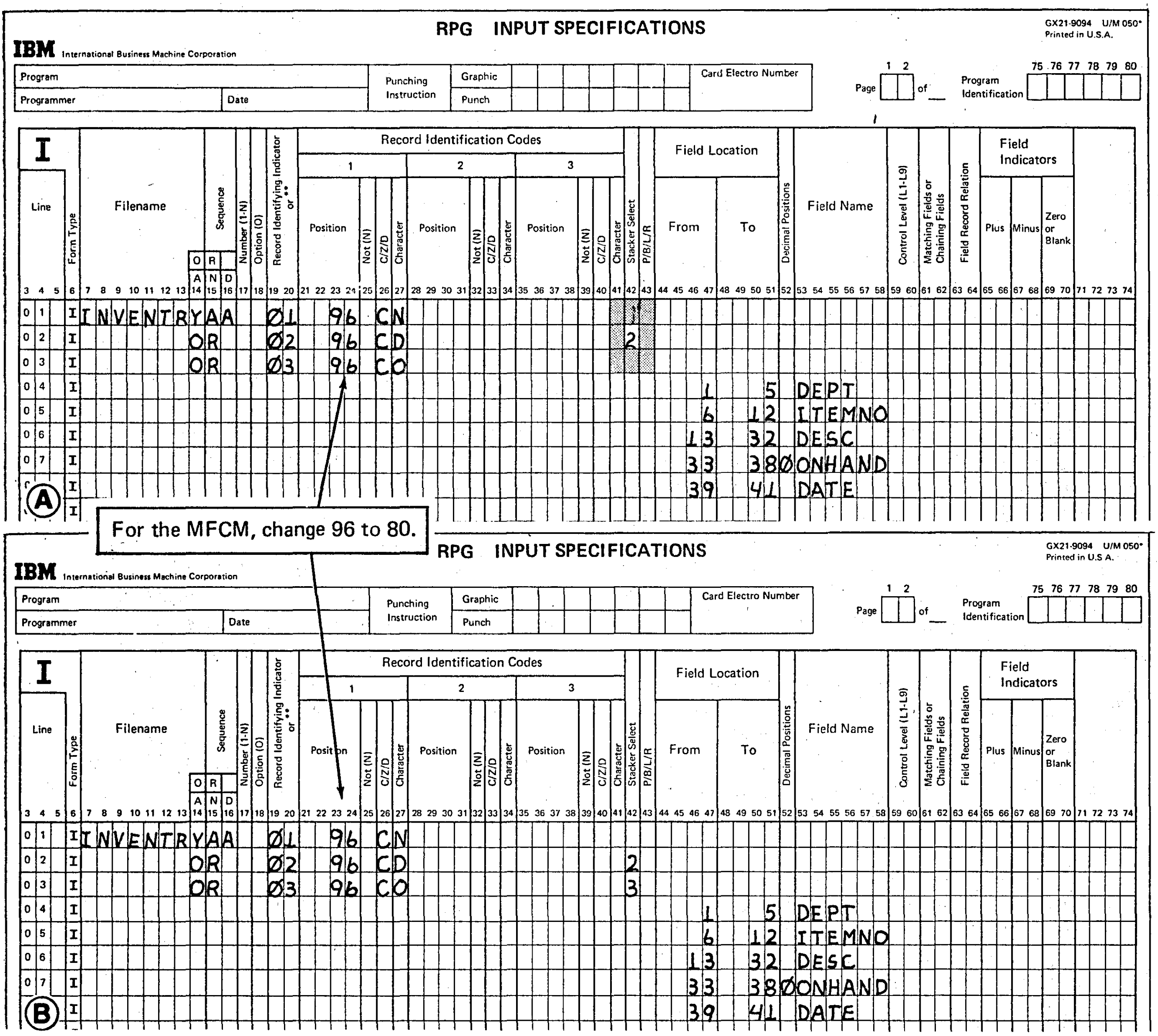
Suppose you want to create a monthly list of all items in a retail store. Three card types are used: one for new items, one for discontinued items, and one for all other items. The manager wants to take all cards describing discontinued items out of his file. He can do this by specifying the stacker into which the card type is to be selected. The specification in Figure 4-17, insert A, line 02, will separate the discontinued item cards (card identified with a D in the last column) from the other cards by putting them in stacker 2.

Notice that the OR relationship was used in describing the three card types. When stacker selection is done with the OR relationship one rule must be kept in mind: each card type will fall into the stacker indicated for it. For example, Figure 4-17, insert A, shows the stacker selection entry 2 for the second card type. The other card types have no stacker selecton entry. They are, therefore, stacked in stacker 1 if they were entered in the primary hopper or in stacker 4 or 5 if entered in the secondary hopper. According to Figure 4-17, insert B, card type 02 falls into stacker 2, card type 03 falls into stacker 3. Where does card type 01 fall? It will fall into either stacker 1, 4, or 5 depending upon the device used and the hopper in which the file was entered.

Selecting on a Basis Other Than Card Type

Suppose you wish to stacker select input file cards on some basis other than card type, such as the result of calculations, the results of matching records, or the contents of an input field. For example, assume that the cards which contain information concerning new, discontinued, and available items in the store also contain the amount on hand at the end of the month. In addition to listing all items, records of items which need to be reordered are selected to a separate stacker. The critical reorder point occurs when there are 25 items or less left on hand. (This, of course, does not apply to discontinued items.) Thus, in the calculations, ONHAND is always compared to 25. If the amount is equal or less than 25, the item needs to be reordered. All cards describing items to be reordered are to be separated from the others in the file by stacking them in a special stacker.

Where would you specify the stacker select entry that would do this? There are only two possibilities — Input sheet or Output-Format sheet. Remember that input cards can be stacker selected on the Input sheet on the basis of card type only. In our example, not all cards of any one type will describe items that need to be reordered. Therefore, the selection is not on the basis of card type and cannot be specified for an input card on the Input sheet. This leaves only the Output-Format sheet on which to specify this stacker selection. But since our file is not an output file, how could it be specified on the Output-Format sheet? Remember that a combined file serves for both input and output. Therefore, a file from which cards are to be stacker selected must be defined as a *combined file.*



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	Stacker Select	From	To	Decimal Positions	Field Name
01	I	INVENTRY	AA			Ø1	96		C	N	1				
02	I		OR			Ø2	96		C	D	2				
03	I		OR			Ø3	96		C	O					
04	I											1	5		DEPT
05	I											6	12		ITEMNO
06	I											13	32		DESC
07	I											33	38	Ø	ONHAND
	I											39	41		DATE
	I														

Ⓐ

For the MFCM, change 96 to 80.

IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	Stacker Select	From	To	Decimal Positions	Field Name
01	I	INVENTRY	AA			Ø1	96		C	N					
02	I		OR			Ø2	96		C	D	2				
03	I		OR			Ø3	96		C	O	3				
04	I											1	5		DEPT
05	I											6	12		ITEMNO
06	I											13	32		DESC
07	I											33	38	Ø	ONHAND
	I											39	41		DATE

Ⓑ

Figure 4-17. Stacker Selecting Cards in an OR Relationship

Figure 4-18 shows the entries which are necessary to stacker select all cards (except discontinued) cards) which contain 25 or less in the ONHAND field. The file would be described as a combined file by placing a C in column 15 of the File Description sheet.

Figure 4-18, insert A, shows that the ONHAND field is compared with 25. If the compare is equal or less, indicator 10 turns on to indicate that the item should be ordered. Indicator 10 is then used on the Output-Format sheet to condition the stacker selection entry (Figure 4-18, insert B, line 01). When 10 is on (there are 25 or less of the item left) and the card is not a discontinued item (indicator 02 is not on), the card is selected into stacker 2. All other cards go into stacker 1.

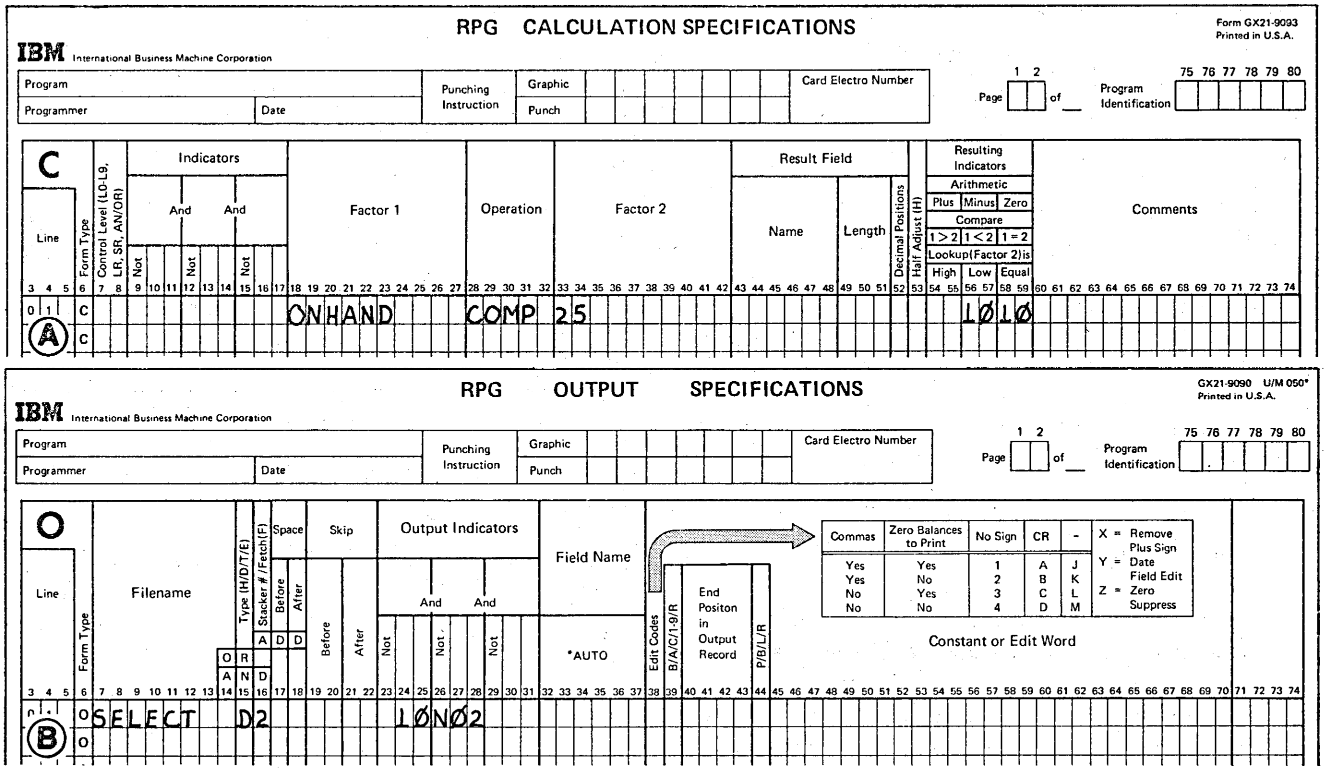
Rules for Stacker Selecting Cards from a Combined File

Combined file cards can be stacker selected by both input and output-format specifications. Input stacker selection is based on card type alone; output stacker selection can be on any other basis. In fact, you can stacker select some card types by input specifications and others by output-format specifications. However, one card type should not have both types of stacker selection specifications. If it does, the input entry is ignored. Furthermore, if you are punching or printing on combined file cards that are also to be stacker selected, the stacker selection entry *must* be on the Output-Format sheet.

Stacker Selecting Output File Cards

Output file cards are stacker selected by output specifications. For example, stacker selection by output specifications can be made on the basis of results of calculations, matching records, content of fields, and error conditions. Output stacker selection can be based on card type, but card type selection is usually done with input specifications.

Consider an end-of-the-month inventory program that: (1) finds the balance on hand for each item, (2) determines if and when an item should be reordered; and (3) finds any items that are overstocked.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Factor 1	Operation	Factor 2	Resulting Indicators Low (56-57)	Resulting Indicators Equal (58-59)
01	C	ONHAND	COMP	25	1Ø	1Ø
	C					

A

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Stacker # / Fetch (F)	Output Indicators
01	O	SELECT	D	2	1Ø NØ2
	O				

B

Figure 4-18. Stacker Selection on the Basis of Calculations

Two files are used: an input file and an output file. The input file contains three types of cards (Figure 4-19):

1. Inventory Balance cards, which contain the number in stock at the beginning of the month, and the maximum and minimum quantities which should be kept in stock.

2. Issue summary cards, which contain the total number issued during each week. Since this is an end of the month job, there may be several of these.

3. Receipt cards, which contain the number added to stock through reorder.

Each item must have a balance card. The other two cards are optional. The output file contains blank cards.

For each item, the total number in stock will be calculated from information on the input cards and then punched into a blank output file card along with all other fields found on the balance card. The format of the output card must be the same as the input balance card, since this output card is used as the balance card for next month's inventory.

Before the balance on hand is punched into the blank cards, the amount is compared to the maximum and minimum quantities in stock established for each item to determine reorder procedure. As a result of the comparison, four conditions could occur:

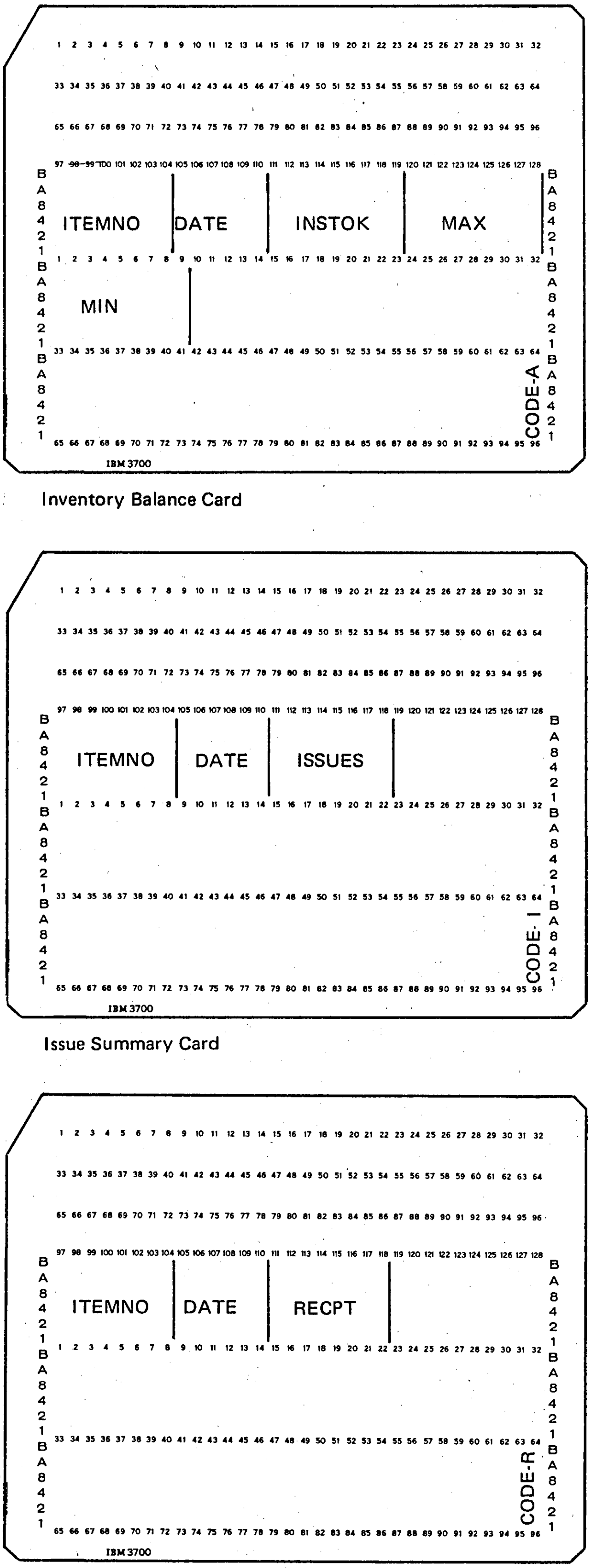
1. If the amount in stock is zero or back-ordered, the item should be reordered immediately.

2. If the amount is greater than zero but equal to or below the minimum, normal reordering procedures should be followed.

3. If the amount is between maximum and minimum, the item does not need to be reordered.

4. If the quantity is greater than the maximum, the manager should be notified of the overstocked item.

Instead of putting all newly punched balance cards into one stacker, it would be more convenient, when preparing to reorder, if they were stacked into different stackers: one stacker for items requiring immediate attention (reorder immediately or greatly overstocked), one for items to be reordered normally, and one for items which need not be reordered. To separate them, you specify different stackers they could go into on the basis of the amount in the INSTOK field.



Inventory Balance Card

Issue Summary Card

Receipt Card

Figure 4-19. Cards Used for Inventory Job

From which file do cards come that are to be stacker selected? They come from the output file since they are the newly created balance cards. Therefore stacker selection entries must be made on the Output-Format sheet. Stacker selection entries are on the basis of the compare operation. Thus these compare operations must set indicators which will be used on the Output-Format sheet to indicate into which stacker cards must fall. Figure 4-20 shows the program specifications.

Figure 4-20, insert A, lines 01, 02, shows operations used for finding the number in stock. When a control break occurs (a card for a new item is read) the number in stock is compared as many as three times (lines 04-06) to determine into which stacker the cards should fall for reordering purposes. Resulting indicators are set off (line 03), since improper selection could otherwise occur due to multiple indicators set for a single condition.

INSTOK is first compared to zero. If it is equal or below, indicator 10 is turned on. If INSTOK is greater than zero, it is compared to the minimum quantity. When INSTOK is equal to or less than minimum, indicator 20 is turned on. If INSTOK is greater than minimum, it is then compared to maximum. Indicator 10 is turned on if INSTOK is greater than maximum; indicator 30 is turned on if it is equal or less.

Any of these indicators can be set: 10, 20, 30. Since they indicate the amount in the field INSTOK, they also indicate into which stacker cards should fall. (See Figure 4-20, insert B.) If 10 is on (INSTOK is zero or less or greater than maximum) cards go into stacker 4. If 20 is on (INSTOK is greater than zero, but equal to or less than the minimum), cards go into stacker 2. If 30 is on (INSTOK is greater than MIN but less than or equal to MAX) cards go into stacker 3. In all cases, five fields and a constant are punched punched on each blank card before it is stacked.

Rules for Stacker Selecting Cards From An Output File

If no stacker selection entry is made for the output file, cards automatically fall into predefined stackers — stacker 1 if the file is in the primary hopper; stacker 4 if the file is in the secondary hopper. You may, however, cause cards to be stacked in any of the four (or five) available stackers merely by entering 1, 2, 3, 4 or 5 (5 for MFCM Model A1) in column 16 of the Output-Format sheet.

All cards from the same file will fall into the specified stacker unless indicators are used in columns 23-31. If, as in this example, you want cards to fall into different stackers, you *must* use indicators set by calculation operations to condition the stacker selection specifications.

Merging Input and Output File Cards

Putting cards from two different files into the same stacker is known as merging cards. Any two files can be merged. To do so, you merely specify the same stacker for each file.

When input and output file cards are merged, the output file card for *each* program cycle is stacked in front of the input file card read at the beginning of the cycle. Why are output cards stacked before input file cards? According to the way the MFCU and MFCM operate:

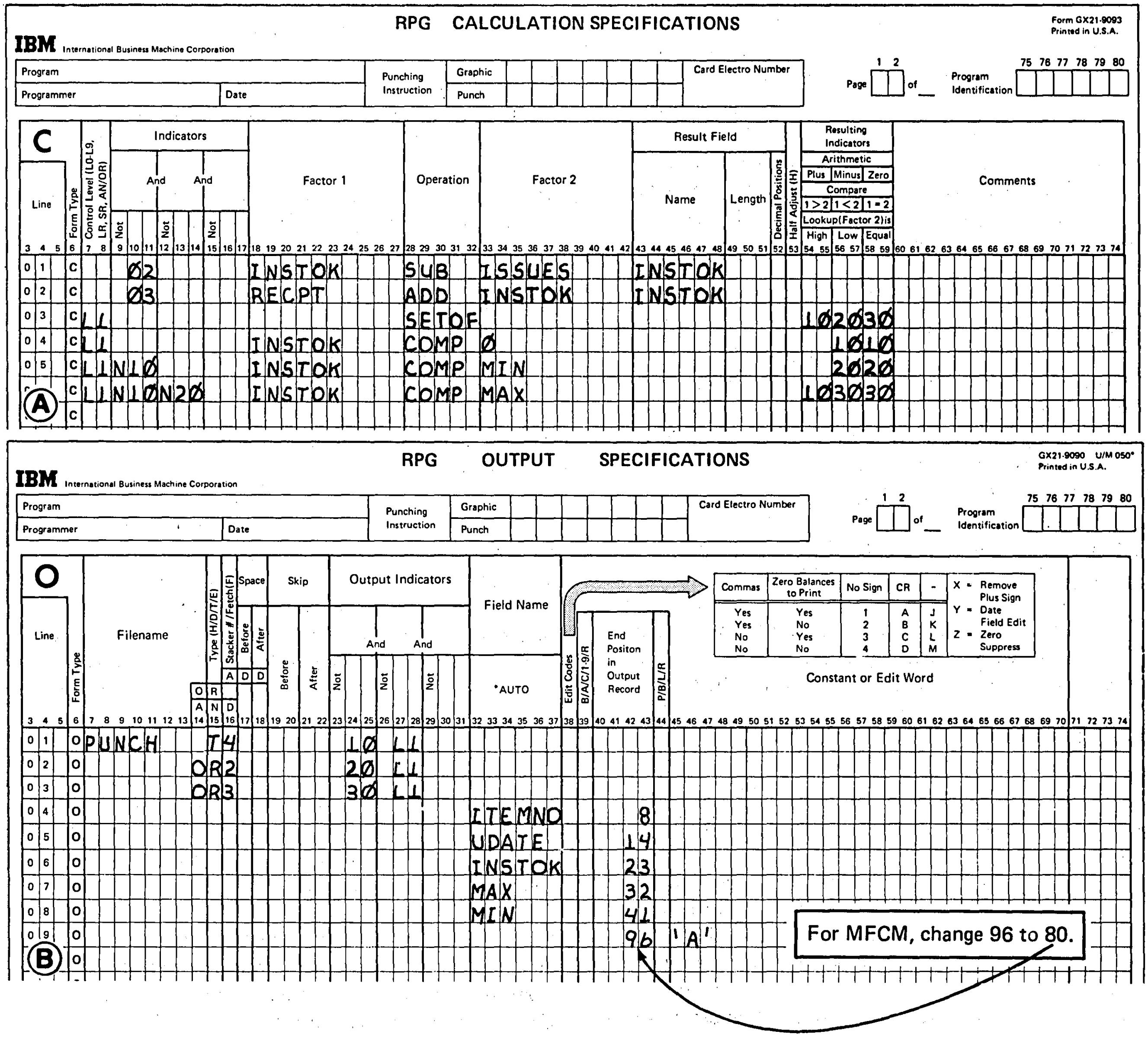
1. An input card is not stacked until the next input card is read.

2. Any output card that is punched is stacked immediately.

These statements are the key to the order of merging. Consider what this means during a regular program cycle. When an output file card is punched, it is stacked. This could be either at total time or detail time. However, the input card read at the beginning of the cycle is not stacked until the next card is read. This results in the output card being stacked in front of the input card. Most often, however, you would want the merged cards ordered so that input cards precede output cards.

Reversal of the normal stacking order can be accomplished by specifying look ahead fields or dual input areas for the input file. When either of these is specified, input cards are stacked before output cards.

Stacker selection cannot be specified for input files with dual input areas. Therefore, if you wish to merge cards from the input and output file, you must merge the cards into the default stacker for the input file (defined previously — see *Stacker Selection*).



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification
Programmer | Date

C

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	And	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
01	C		Ø2			INSTOK	SUB	ISSUES	INSTOK							
02	C		Ø3			RECPT	ADD	INSTOK	INSTOK							
03	C	L1					SETOF						1Ø	2Ø	3Ø	
04	C	L1				INSTOK	COMP	Ø						1Ø	1Ø	
05	C	L1	N1Ø			INSTOK	COMP	MIN						2Ø	2Ø	
Ⓐ	C	L1	N1Ø	N2Ø		INSTOK	COMP	MAX					1Ø	3Ø	3Ø	
	C															

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification
Programmer | Date

O

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E) / OR / AND	Stacker # / Fetch (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	And	And	Field Name / *AUTO	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	PUNCH	T	4					1Ø	L1							
02	O		OR	2					2Ø	L1							
03	O		OR	3					3Ø	L1							
04	O											ITEMNO			8		
05	O											UDATE			14		
06	O											INSTOK			23		
07	O											MAX			32		
08	O											MIN			41		
09	O														96		'A'
Ⓑ	O																

For MFCM, change 96 to 80.

Figure 4-20. End-of-Month Inventory Job

Punching and Printing on Cards

1. Why is it important to print on a card the same information punched in a card?

2. Using the following information, write the output-format specifications to punch and print the following fields on output cards:

Field	*End Position*
CUSTNO	5
NAME	26
AMT	32
INVNO	38
DATE	44
2 (constant)	96 (use 64 for MFCM)

The information should be printed in the same relative positions as it was punched. Do not use *PRINT for this. For the MFCM (Model 15 only), use print head 1 for all fields.

3. Do problem 2 using *PRINT.

Using One File For Both Input and Output

4. When should a file be specified as combined?

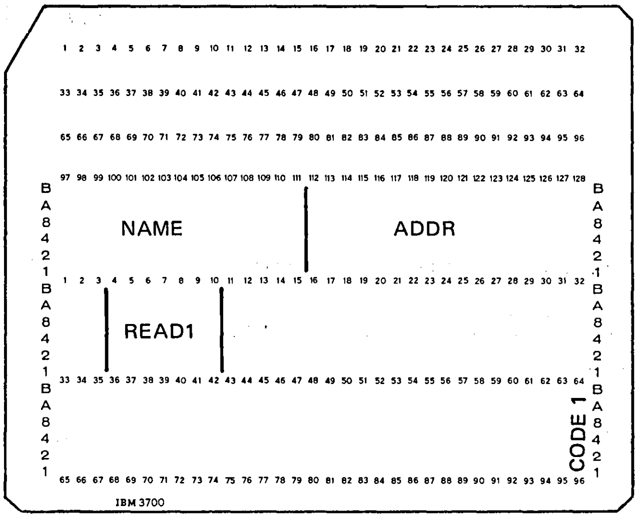
5. If all master cards in a file are to be separated from item cards in the same file, the file type should be specified as______.

6. True or False? Both printer files and card files can be combined files.

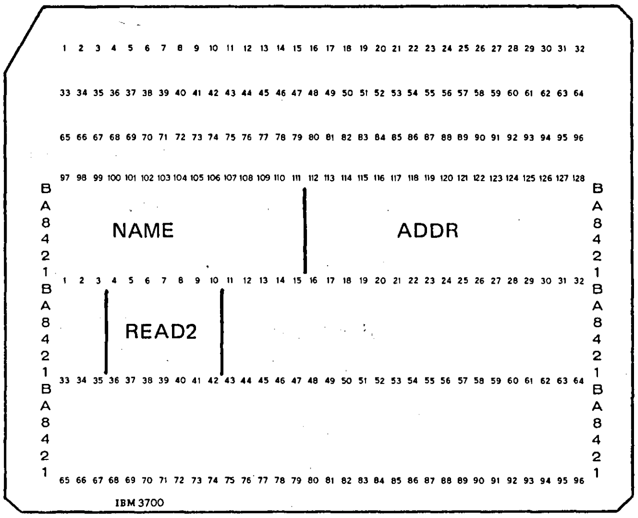
7. An electric company wishes to find the amount each customer owes for the electricity he used during the past month. The input file consists of three types of records for each customer:

- READ1 which contains the meter reading at the beginning of the month.
- READ2 which contains the meter reading at the end of the month. This card will be used as next month's READ1 card. It now contains a blank in the last column, and must therefore be punched with a 1 in the last column.
- AMOUNT DUE which contains a blank field (AMTDUE) which will be punched with the amount each customer owes.

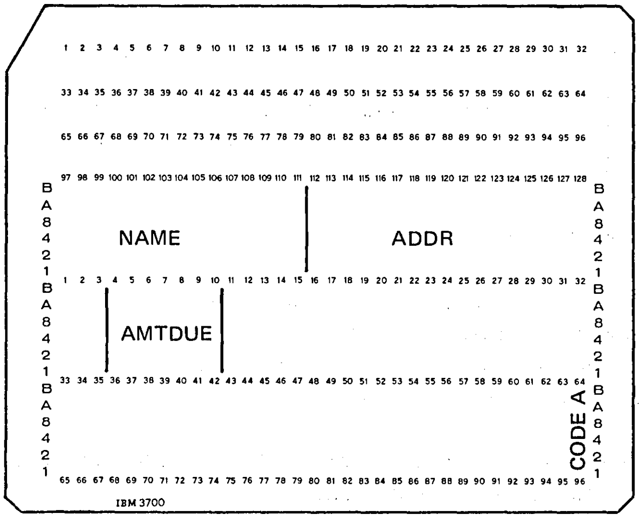
For each account these three cards must be present and in the order indicated.



READ1 Card



READ2 Card



Amount Due Card

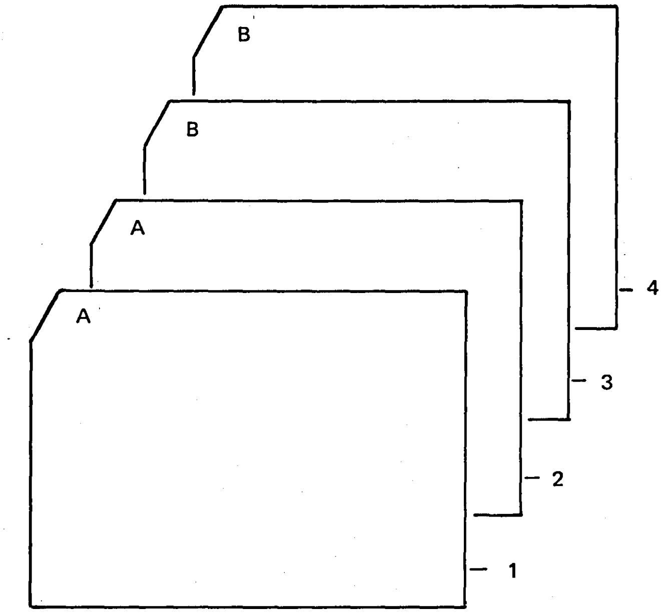
The program must:

- Find the number of kilowatt hours (KWH) of electricity used during the month. The reading has two decimal places.
- Multiply the number of kilowatt hours used by rate per kilowatt hour to find amount due. (AMTDUE has 2 decimal places.) Rate is $.05 per KWH for the first 50 KWH, then $.02 per KWH for all over the first 50 KWH.
- Punch the amount due on the appropriate card.
- Separate all three card types into different stackers.

Make the necessary entries on the File Description, Input, Calculation, and Output-Format sheets.

Stacker Selection

8. If stacker selection is specified on the Output-Format sheet during detail output for card 3, which card will be selected? Which card will be selected if stacker selection is specified during total time output after control group A?



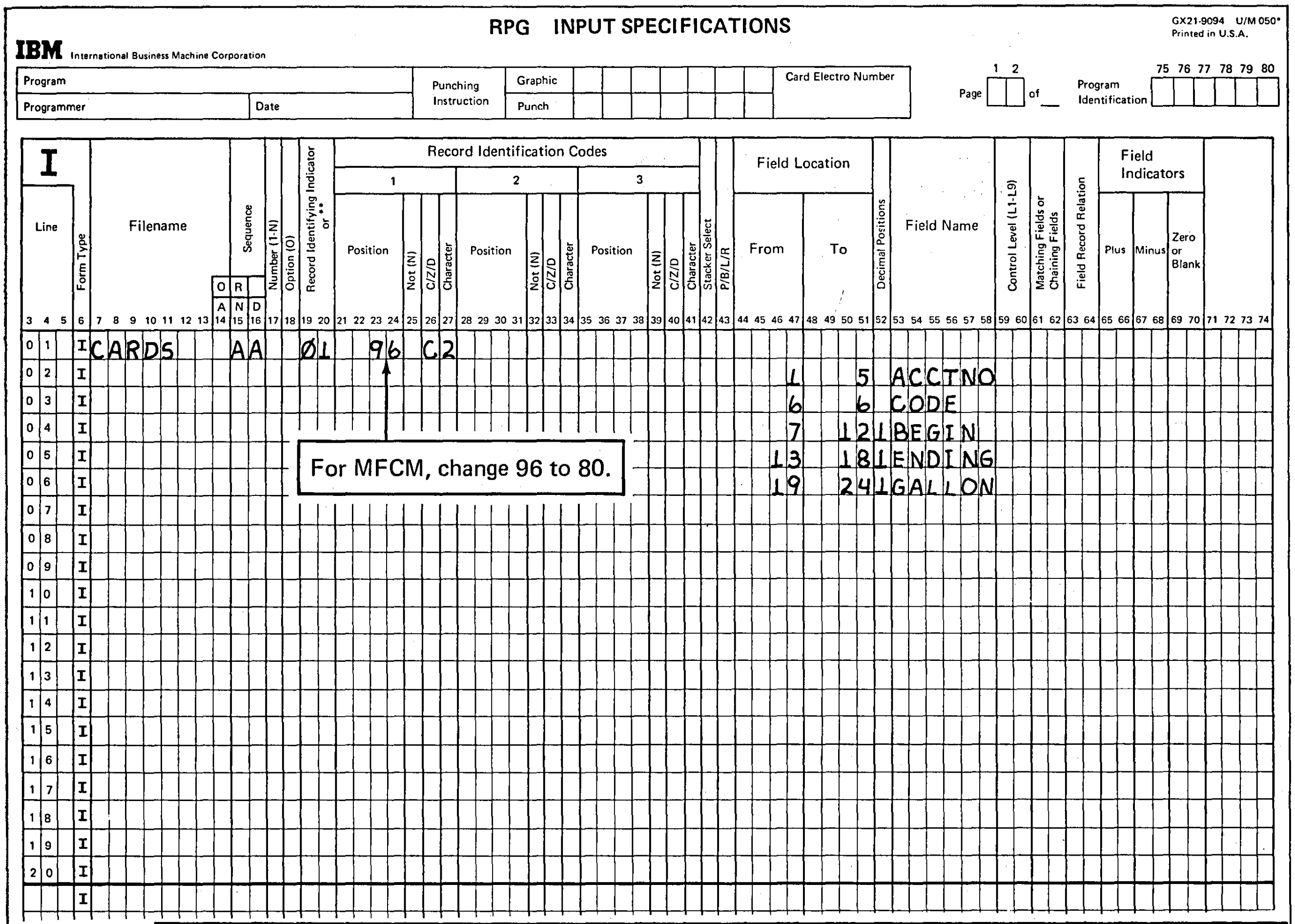
9. At each stop they make, drivers working for a fuel oil company record beginning and ending meter readings and the number of gallons of oil delivered to the customer. Later the account number, meter readings, and gallons delivered are punched into cards.

All regular customers are charged 15¢ per gallon. However hospital and government agencies receive a 2% discount. The code to show which customers receive a discount is in the account field. If the last digit is 0, no discount is given; but if the last digit is 5, the discount is given.

Write the calculation and output-format specifications to:

a. Check the driver's calculations in determining gallons delivered to each account by subtracting beginning meter reading from ending meter reading.
b. Calculate the amount charged to each account (AMOUNT).
c. Find total number of gallons sold for the day (TOTALG) and total amount charged (TOTALA).
d. Print a report listing daily transactions and totals. If there is an error in driver's calculations, print the account number, code, and a message, 'CALCULATION ERROR'.
e. Stacker select cards for customers receiving discounts into stacker 2. All others go into stacker 3.

The input specifications and the layout of the printed report are as follows:



IBM International Business Machine Corporation

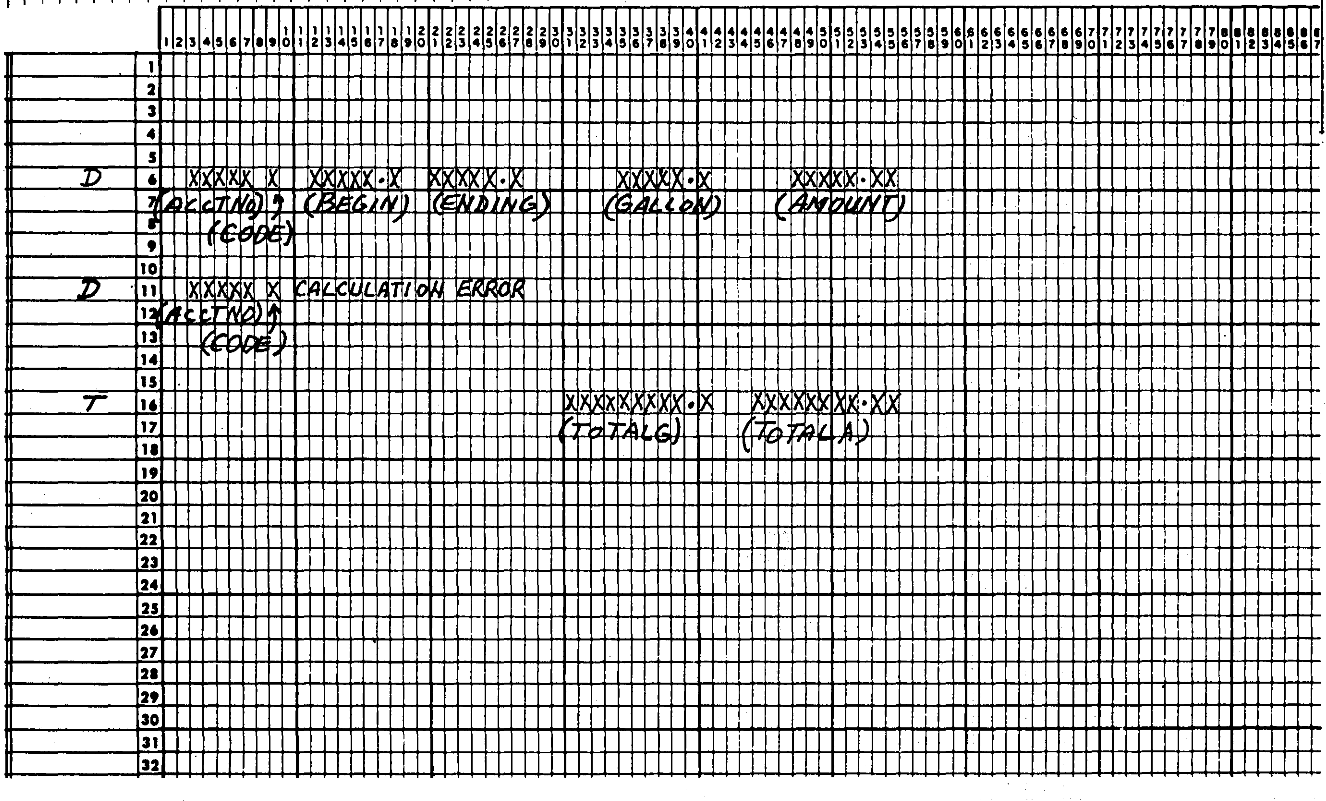
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program ____ Programmer ____ Date ____ Punching Instruction: Graphic ____ Punch ____ Card Electro Number ____ Page ____ of ____ Program Identification ____

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	From	To	Decimal Positions	Field Name
01	I	CARDS	AA			Ø1	96		C	2				
02	I										1	5		ACCTNO
03	I										6	6		CODE
04	I										7	12	1	BEGIN
05	I										13	18	1	ENDING
06	I										19	24	1	GALLON
07	I													
08	I													
09	I													
10	I													
11	I													
12	I													
13	I													
14	I													
15	I													
16	I													
17	I													
18	I													
19	I													
20	I													
	I													

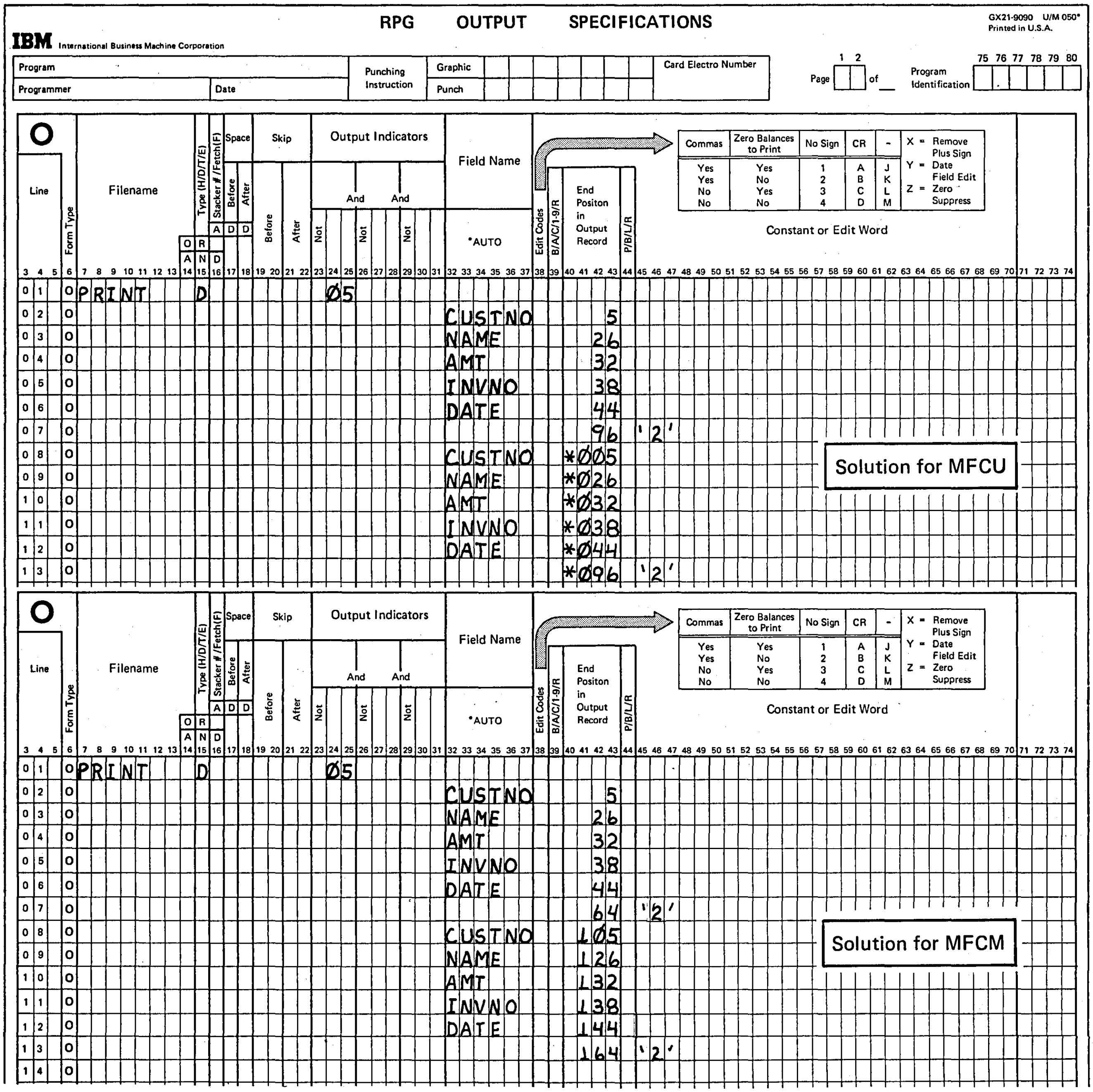
For MFCM, change 96 to 80.



```
      1234567890123456789012345678901234567890123456789012345678901234567890
D   6   XXXXX X  XXXXX.X  XXXXX.X       XXXXX.X      XXXXX.XX
    7 (ACCTNO)  (BEGIN)  (ENDING)     (GALLON)     (AMOUNT)
    8      (CODE)
D  11   XXXXX X CALCULATION ERROR
   12 (ACCTNO)
   13     (CODE)
T  16                            XXXXXXXXX.X   XXXXXXXX.XX
   17                           (TOTALG)      (TOTALA)
```

1. Printing the same information that is punched on a card:
   a. Enables you to easily see what is on the card. You don't have to analyze each punch combination.
   b. Makes it easier to recreate a card that is damaged to the extent that it cannot be processed.

2.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program

Programmer

Date

Punching Instruction | Graphic | Punch

Card Electro Number

Page of

Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

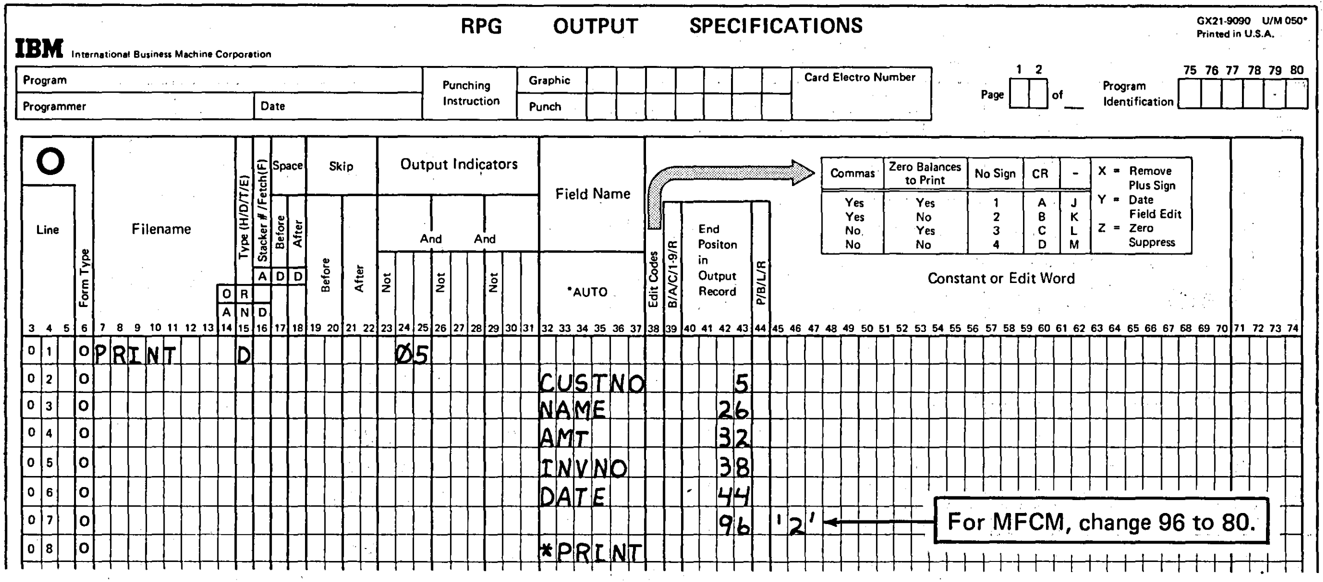
Solution for MFCU

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	D	Ø5			
02	O				CUSTNO	5	
03	O				NAME	26	
04	O				AMT	32	
05	O				INVNO	38	
06	O				DATE	44	
07	O					96	'2'
08	O				CUSTNO	*ØØ5	
09	O				NAME	*Ø26	
10	O				AMT	*Ø32	
11	O				INVNO	*Ø38	
12	O				DATE	*Ø44	
13	O					*Ø96	'2'

Solution for MFCM

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	D	Ø5			
02	O				CUSTNO	5	
03	O				NAME	26	
04	O				AMT	32	
05	O				INVNO	38	
06	O				DATE	44	
07	O					64	'2'
08	O				CUSTNO	1Ø5	
09	O				NAME	126	
10	O				AMT	132	
11	O				INVNO	138	
12	O				DATE	144	
13	O					164	'2'
14	O						

3.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	D	Ø5			
02	O				CUSTNO	5	
03	O				NAME	26	
04	O				AMT	32	
05	O				INVNO	38	
06	O				DATE	44	
07	O					96	'2'
08	O				*PRINT		

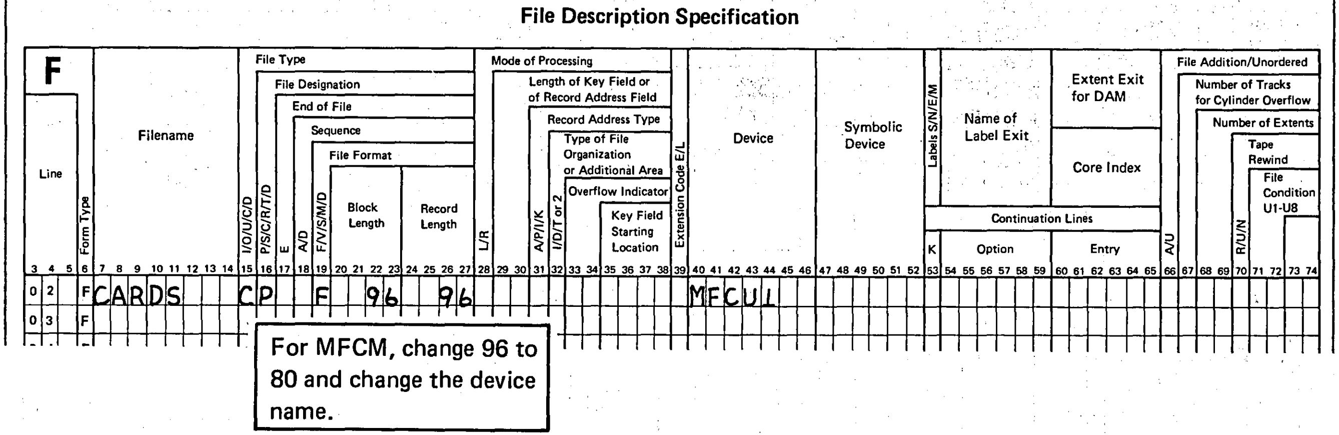
For MFCM, change 96 to 80.

4. A file should be specified as combined if it is both read and punched or if cards from the file are stacker selected on a basis other than card type.

5. Input (stacker selection is on basis of record type here.)

6. False, only card files can be designated as combined files.

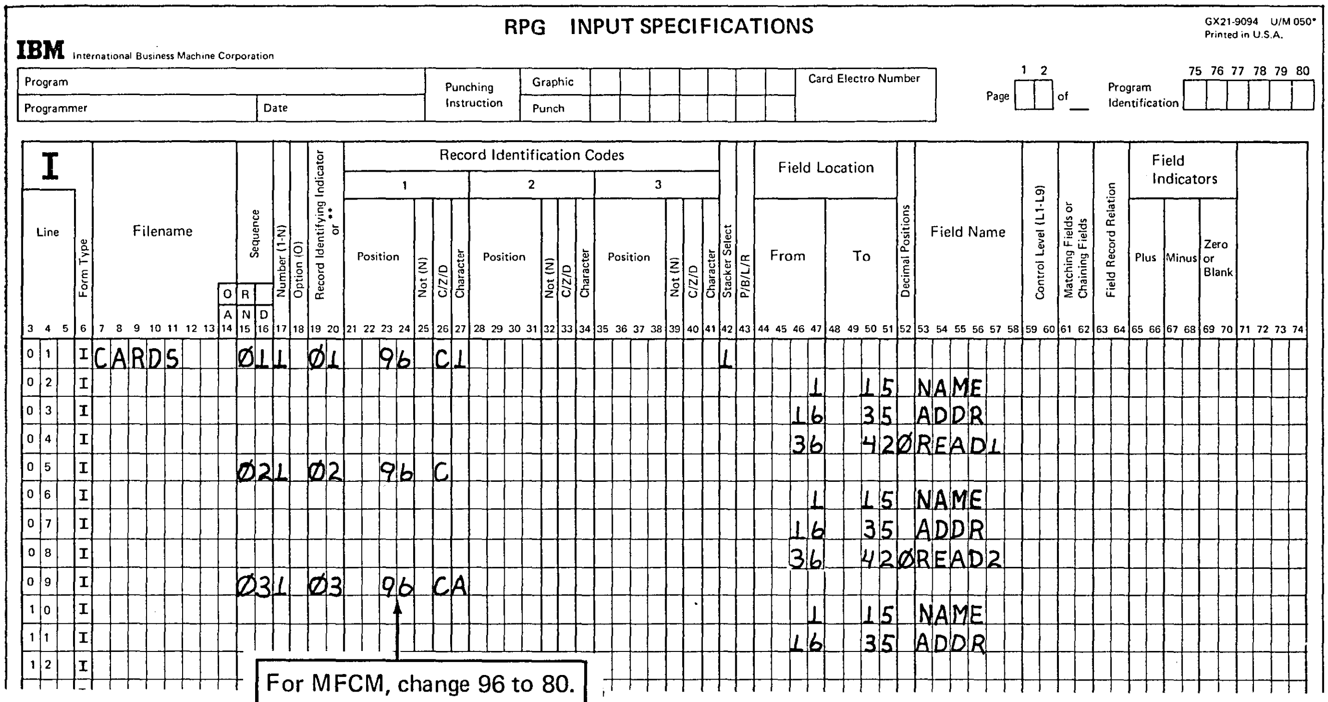
7.



File Description Specification

Line	Form Type	Filename	File Type	File Designation	File Format	Block Length	Record Length	Device
02	F	CARDS	C	P	F	96	96	MFCU1
03	F							

For MFCM, change 96 to 80 and change the device name.



IBM International Business Machine Corporation

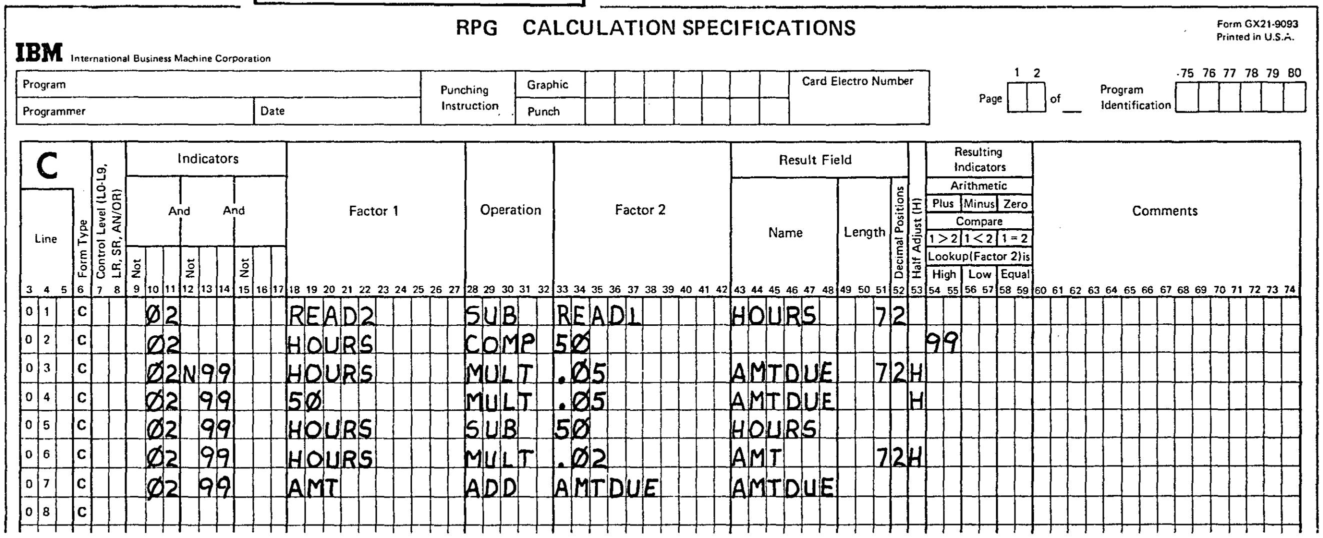
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position (1)	Not (N)	C/Z/D	Character	Stacker Select	From	To	Decimal Positions	Field Name	Control Level	Matching Fields	Field Record Relation	Plus	Minus	Zero or Blank
01	I	CARDS	Ø1	1		Ø1	96		C	1	1										
02	I											1	15		NAME						
03	I											16	35		ADDR						
04	I											36	42	Ø	READ1						
05	I		Ø2	1		Ø2	96		C												
06	I											1	15		NAME						
07	I											16	35		ADDR						
08	I											36	42	Ø	READ2						
09	I		Ø3	1		Ø3	96		C	A											
10	I											1	15		NAME						
11	I											16	35		ADDR						
12	I																				

For MFCM, change 96 to 80.



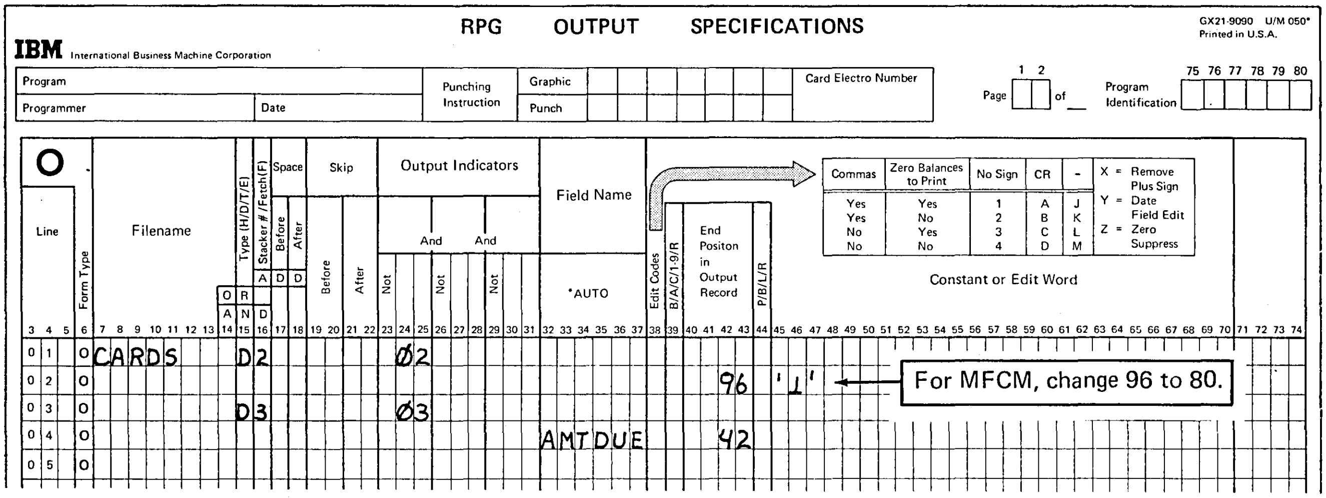
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus / High	Minus / Low	Zero / Equal	Comments
01	C		Ø2	READ2	SUB	READ1	HOURS	7	2					
02	C		Ø2	HOURS	COMP	5Ø					99			
03	C		Ø2 N99	HOURS	MULT	.Ø5	AMTDUE	7	2	H				
04	C		Ø2 99	5Ø	MULT	.Ø5	AMTDUE			H				
05	C		Ø2 99	HOURS	SUB	5Ø	HOURS							
06	C		Ø2 99	HOURS	MULT	.Ø2	AMT	7	2	H				
07	C		Ø2 99	AMT	ADD	AMTDUE	AMTDUE							
08	C													



IBM International Business Machine Corporation — RPG OUTPUT SPECIFICATIONS — GX21-9090 U/M 050* Printed in U.S.A.

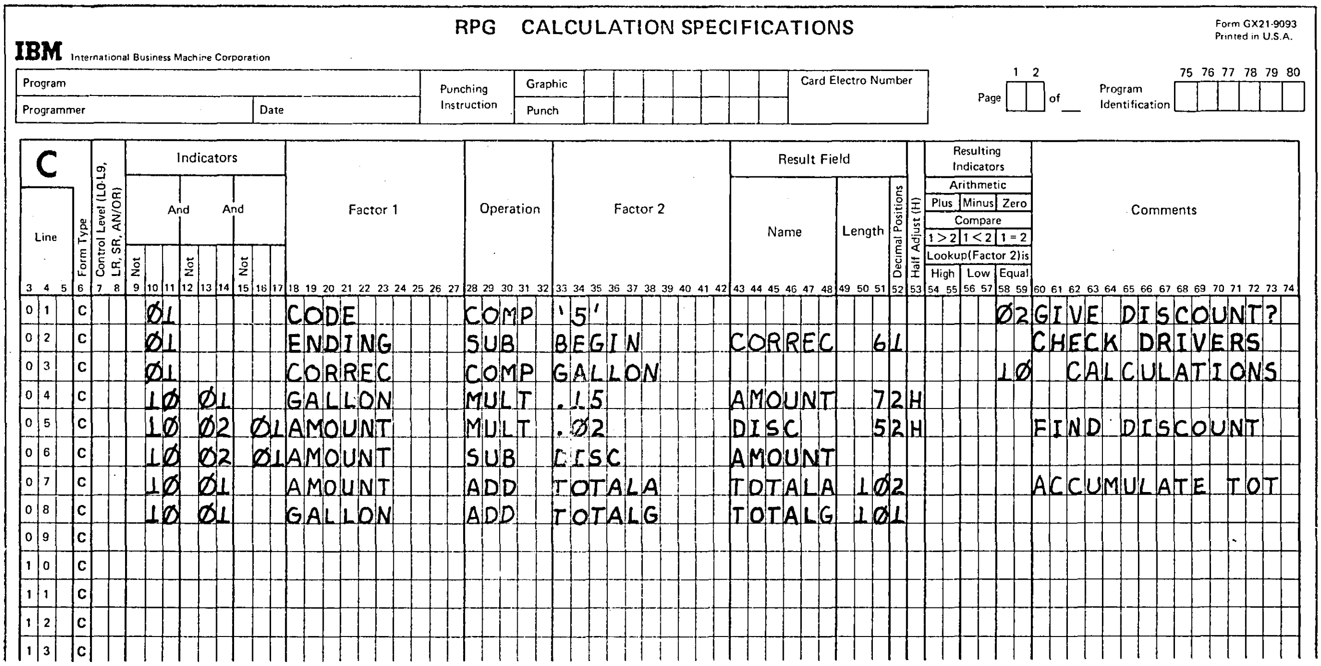
Line	Form Type	Filename	Type (H/D/T/E)	Stacker # / Fetch (F)	Space Before	Space After	Skip Before	Skip After	Not	Output Indicators	Not	And	Not	And	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	CARDS	D	2						Ø2										
02	O																	96		'1'
03	O		D	3						Ø3										
04	O														AMTDUE			42		
05	O																			

For MFCM, change 96 to 80.

Since cards in this file are to be both read and punched the file must be defined as a combined file (C in column 15 of the File Description sheet). The card type identified by a 1 requires no punching, and therefore can be stacker selected on the Input sheet. A 1 was entered in column 42 of the Input sheet to indicate the stacker. Leaving this column blank would also indicate that the card type goes into stacker 1 because cards entered in primary hopper of the MFCU are automatically stacked in stacker 1. Output operations are performed on card types 02 and 03. Stacker selection is, therefore, specified for each on the Output Specifications Sheet in column 16.

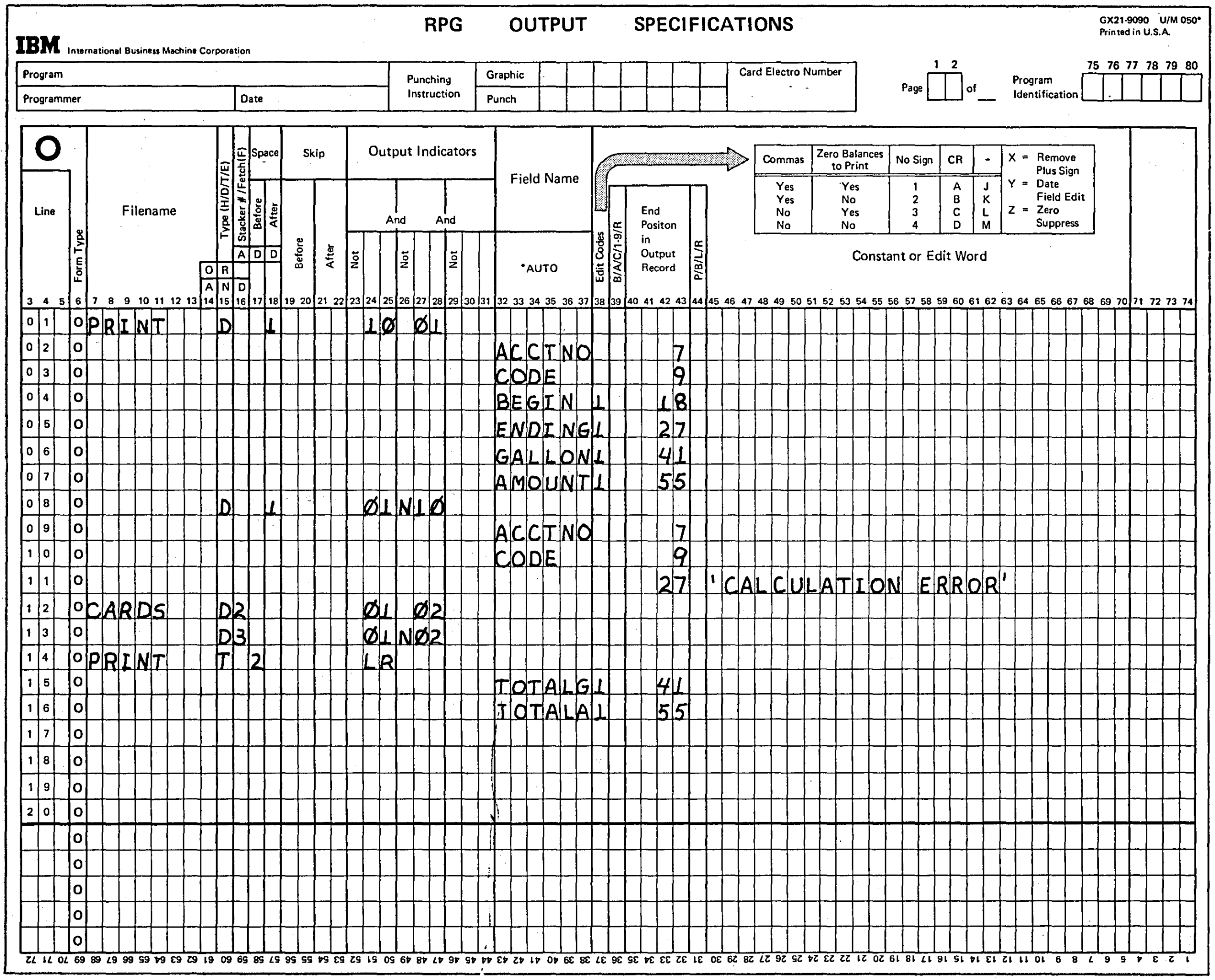
8. a. Card 3 – the card that is being processed.
   b. Card 3 – the card which caused the control break.

9.



IBM International Business Machine Corporation — RPG CALCULATION SPECIFICATIONS — Form GX21-9093 Printed in U.S.A.

Line	Form Type	Control Level	Indicators	And	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus / High	Minus / Low	Zero / Equal	Comments
01	C		Ø1			CODE	COMP	'5'							Ø2	GIVE DISCOUNT?
02	C		Ø1			ENDING	SUB	BEGIN	CORREC	6	1					CHECK DRIVERS
03	C		Ø1			CORREC	COMP	GALLON							1Ø	CALCULATIONS
04	C		1Ø	Ø1		GALLON	MULT	.15	AMOUNT	7	2	H				
05	C		1Ø	Ø2	Ø1	AMOUNT	MULT	.Ø2	DISC	5	2	H				FIND DISCOUNT
06	C		1Ø	Ø2	Ø1	AMOUNT	SUB	DISC	AMOUNT							
07	C		1Ø	Ø1		AMOUNT	ADD	TOTALA	TOTALA	1Ø	2					ACCUMULATE TOT
08	C		1Ø	Ø1		GALLON	ADD	TOTALG	TOTALG	1Ø	1					
09	C															
10	C															
11	C															
12	C															
13	C															



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program		Punching Instruction	Graphic
Programmer	Date		Punch

Card Electro Number

Page □□ of __

Program Identification □□□□□□

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Stacker # / Fetch (F)	Space Before	Space After	Skip Before	Skip After	Not	Ind.	Not	Ind.	Not	Ind.	Field Name	Edit Codes	B/A/C/1-9/R	End Positon in Output Record	P/B/L/R	Constant or Edit Word
01	O	PRINT	D			1				1Ø		Ø1								
02	O														ACCTNO			7		
03	O														CODE			9		
04	O														BEGIN	1		18		
05	O														ENDING	1		27		
06	O														GALLON	1		41		
07	O														AMOUNT	1		55		
08	O		D			1				Ø1	N	1Ø								
09	O														ACCTNO			7		
10	O														CODE			9		
11	O																	27		'CALCULATION ERROR'
12	O	CARDS	D	2						Ø1		Ø2								
13	O		D	3						Ø1	N	Ø2								
14	O	PRINT	T		2					LR										
15	O														TOTALG	1		41		
16	O														TOTALA	1		55		
17	O																			
18	O																			
19	O																			
20	O																			
	O																			
	O																			
	O																			
	O																			
	O																			

You must be certain to check to see if the code is 5 or 0. The resulting indicator showing the result of the compare is then used on the Output-Format sheet to show into which stacker cards should fall. Stacker selection must be specified as a detail operation so that the correct card will be selected. Any report formatting you choose is acceptable.

Chapter 5. Controlling Operations In An RPG II Program

CHAPTER 5 DESCRIBES:

Additional uses of indicators to control calculations and output.

Controlling operations on the basis of the next record in a file.

Manipulating data by moving it from one field to another.

Saving storage space and coding in calculations by using branching and subroutines.

Special uses of control level indicators.

Binary field operations.

Increasing the speed of RPG II operations.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

General usage of the following indicators: 01-99, MR, L1-L9, LR, 1P, OA-OG, OV.

The concept of matching records.

Coding of arithmetic operations in calculations.

RPG II object program cycle.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Control calculations and output using H1-H9, U1-U8, and Resulting Indicators.

Condition calculations using OA-OG and OV indicators.

Use the RPG II look ahead feature.

Code specifications to move data from one field to another.

Branch in calculations using GOTO and TAG.

Employ subroutines in calculations using BEGSR, ENDSR, and EXSR.

Cause an artificial control break and total operations using L0.

Use control level indicators to perform group printing.

Control calculations using binary switches.

State advantages of dual input/output areas and correctly code for them.

Note: You can use the review questions contained in *Review 5* at the end of this chapter to test your comprehension of each topic in the chapter. Questions are grouped according to the topic to which they apply. Answers follow the review questions.

INTRODUCTION

There are many ways that you can control the performance of RPG II operations. You have already learned the basic elements of controlling calculations and output, especially the concept of the RPG II object program cycle and the use of indicators to condition specifications. This chapter supplements those basic concepts by presenting topics that will help you improve the performance of your RPG II programs and do more complex jobs.

ADDITIONAL USES OF INDICATORS TO CONTROL CALCULATIONS AND OUTPUT

On the Calculation and Output-Format sheets, you describe all the calculations and output to be done in your program. Sometimes, all the calculation and output operations must be performed on every program cycle. More often, however, you want operations done only under certain conditions. For example, you may want to perform a calculation or do some output only when a control break occurs, do an operation only when a certain record type is read, or do certain operations only on certain program runs.

In columns 7-17 (Indicators) of the Calculations sheet and columns 23-31 (Output Indicators) on the Output-Format sheet, you can specify when certain calculation and output operations are to be done. Some of the indicators that can be used, and the conditions they signal, are:

Indicator	*Condition*
01-99	Operation is done only when a specific record type has been read, or when the result of a calculation or the contents of a field are as desired.
MR	Operation is done only when records match.
L1-L9	Operation is done only when a control break occurs.
LR	Operation is done after all records have been read and processed.
1P	Output record prints only on the first page.
OA-OG; OV	Output record prints only when overflow occurs.

You are probably somewhat familiar with the use of these indicators from previous education, reading, or other topics in this book. However, there are other kinds and uses of indicators with which you may not be so familiar. This section discusses:

1. Halt indicators used to tell what operations should be done on an error condition.

2. External indicators used to tell what operations should be done for a specific program run.

3. Overflow indicators used to tell what *calculations* should be done when overflow occurs.

In addition to these new uses of indicators, this section also describes conditioning of operations based on the results of certain calculations.

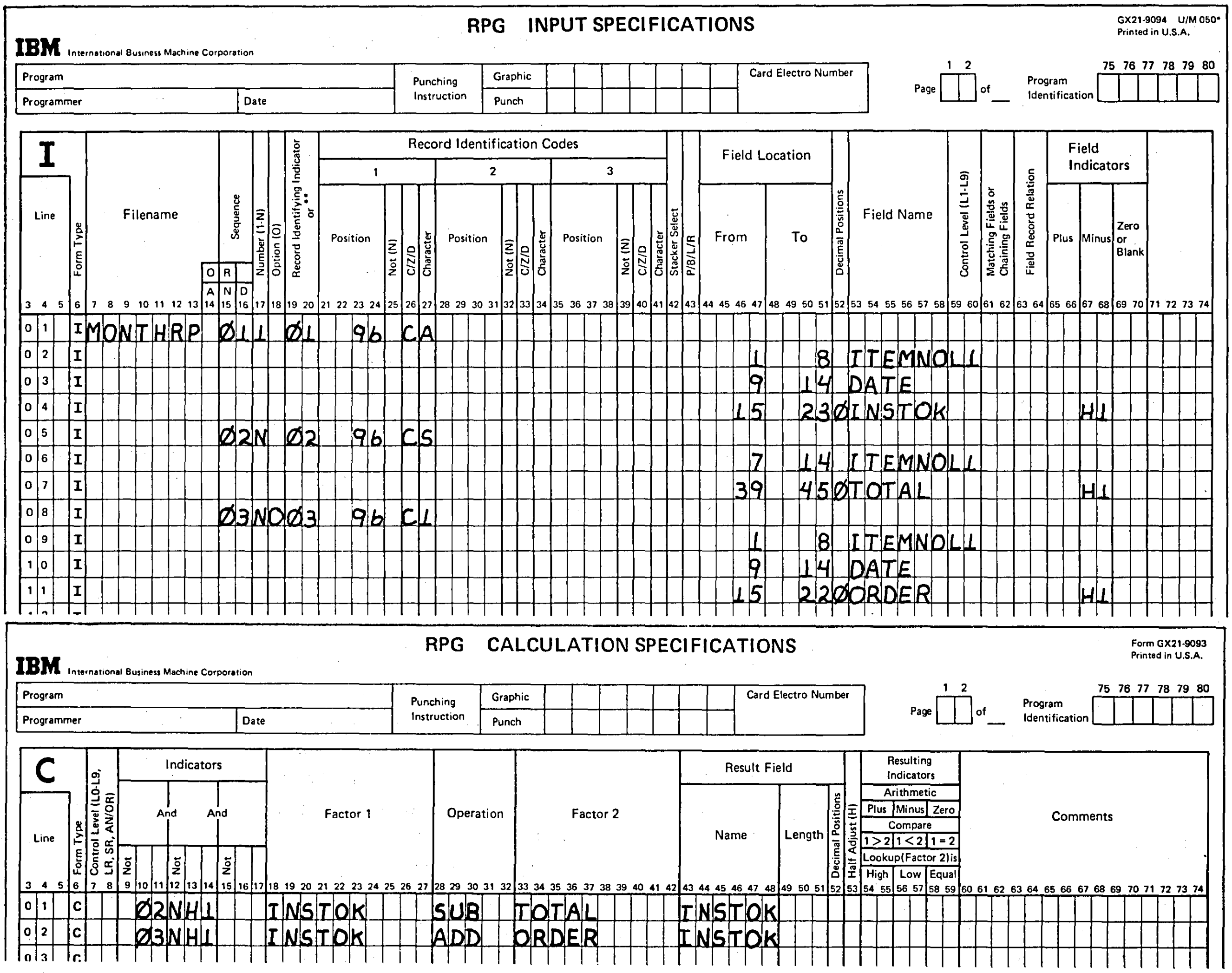
Preventing Operations From Being Done When an Error Occurs

Halt indicators are used to test for an error condition in your data. According to RPG II program logic, a halt does not occur as soon as the error condition is found (as soon as the halt indicator is turned on). Instead, the program cycle is completed before the halt occurs. This means that additional operations may be performed in error unless you specify otherwise.

Preventing Calculations When an Error Occurs

Specifications shown in Figure 5-1 illustrate the use of H1 to prevent calculations. Tests are made to determine if the INSTOK, TOTAL, or ORDER fields on record types 01, 02, or 03, respectively, are negative. A negative value in any of these fields is an error condition. When an error is found, H1 turns on. Since calculations [normally] are done when 02 and 03 record types are read, conditioning these calculations by NH1 prevents them from being done when data is erroneous.

Halt indicators can also be specified on the Calculation sheet to test for an error. For example, in Figure 5-2, H1 is set on if the result of operation in line 01 is negative. If quantity in stock (INSTOK) is negative after quantity shipped (QTYSH) has been subtracted, an error has occurred. H1 turns on and the system will halt after the current cycle.



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	From	To	Decimal Positions	Field Name	Field Indicators Minus
01	I	MONTHRP	Ø1	1		Ø1	96		C	A					
02	I										1	8		ITEMNO11	
03	I										9	14		DATE	
04	I										15	23	Ø	INSTOK	H1
05	I		Ø2	N		Ø2	96		C	S					
06	I										7	14		ITEMNO11	
07	I										39	45	Ø	TOTAL	H1
08	I		Ø3	N	O	Ø3	96		C	1					
09	I										1	8		ITEMNO11	
10	I										9	14		DATE	
11	I										15	22	Ø	ORDER	H1

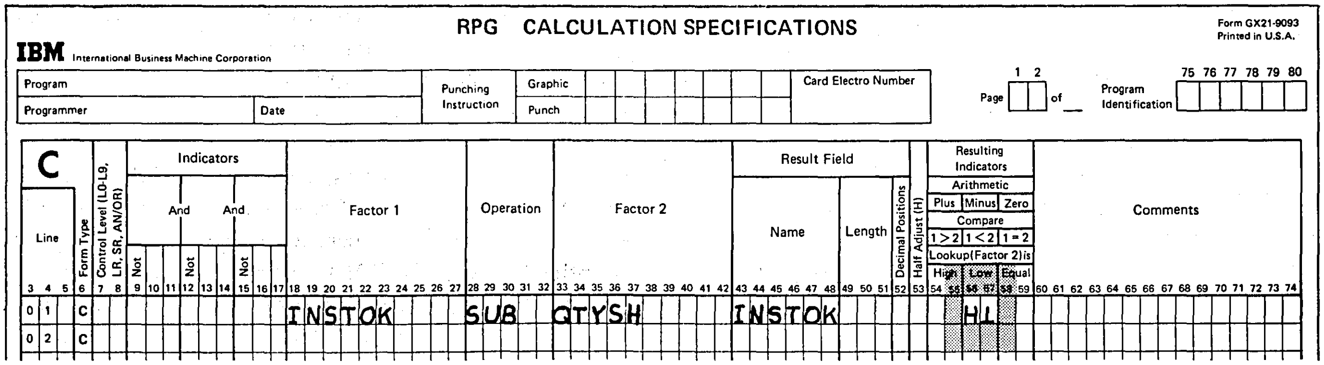
RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name
01	C	Ø2 N H1	INSTOK	SUB	TOTAL	INSTOK
02	C	Ø3 N H1	INSTOK	ADD	ORDER	INSTOK

Figure 5-1. Conditioning Calculations by a Halt Indicator



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Factor 1	Operation	Factor 2	Result Field Name	Resulting Indicators Low
01	C	INSTOK	SUB	QTYSH	INSTOK	H1
02	C					

Figure 5-2. Testing Result Field for Error Conditions

Preventing an Entire Record From Being Written

Figure 5-3, line 07, shows an output operation conditioned so that the record specified will be written only when the halt indicator is not on (NH1). When the halt indicator is on (H1), it will be bypassed.

Preventing Fields From Being Written

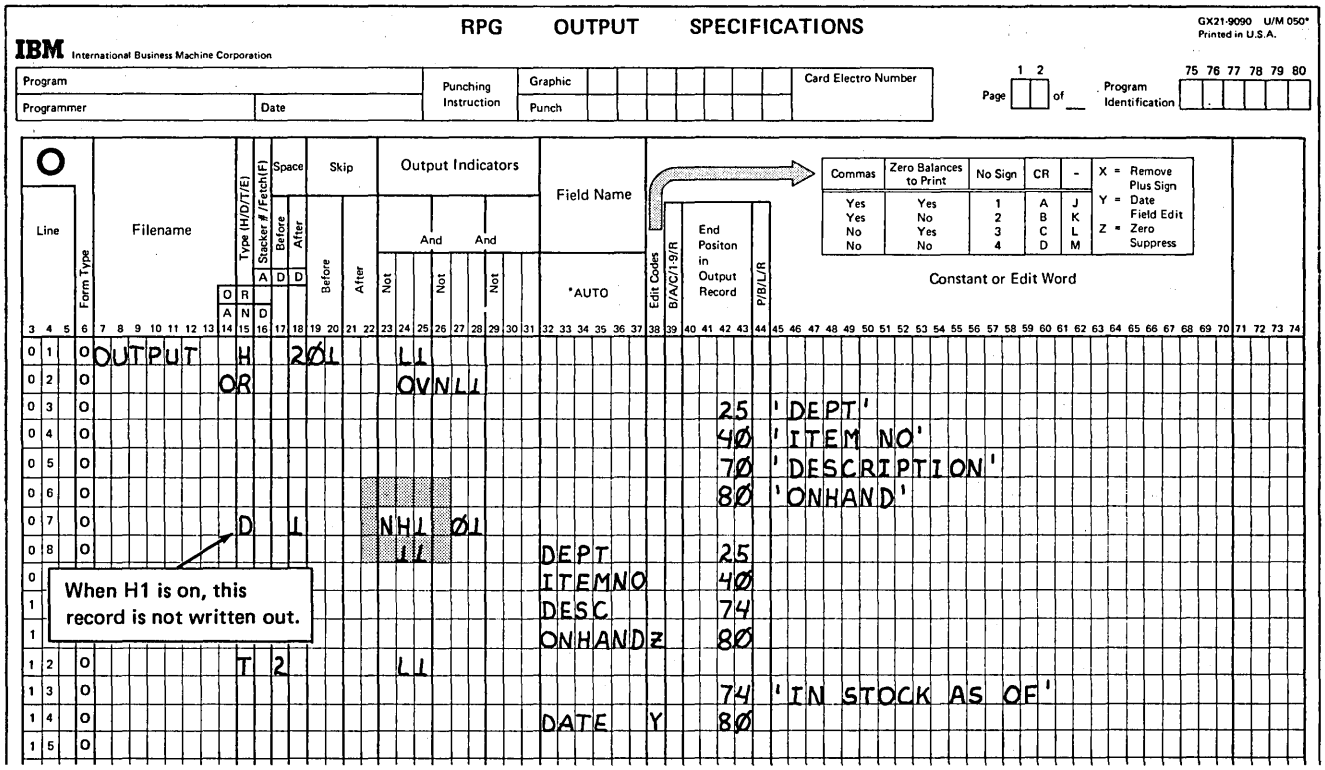
Suppose, however, that you do not want to bypass the writing of the entire record, but want some fields written even when a halt condition occurs. For this case you should use the halt indicator to condition certain fields within the record instead of conditioning the entire record. This way, when an error occurs, some fields will be written and some will not.

Figure 5-4 shows the specifications which will bypass the writing of all fields except DEPT and ITEMNO when an error occurs.

Doing Output Only When an Error Occurs

It is also possible to condition records or fields so that they are written only when an error condition occurs. Figure 5-5 shows the specifications which do this.

Using the halt indicator will cause the computer to stop after all operations are completed for the record causing the error. You may restart processing immediately, however, by pressing the start button on the processing unit.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

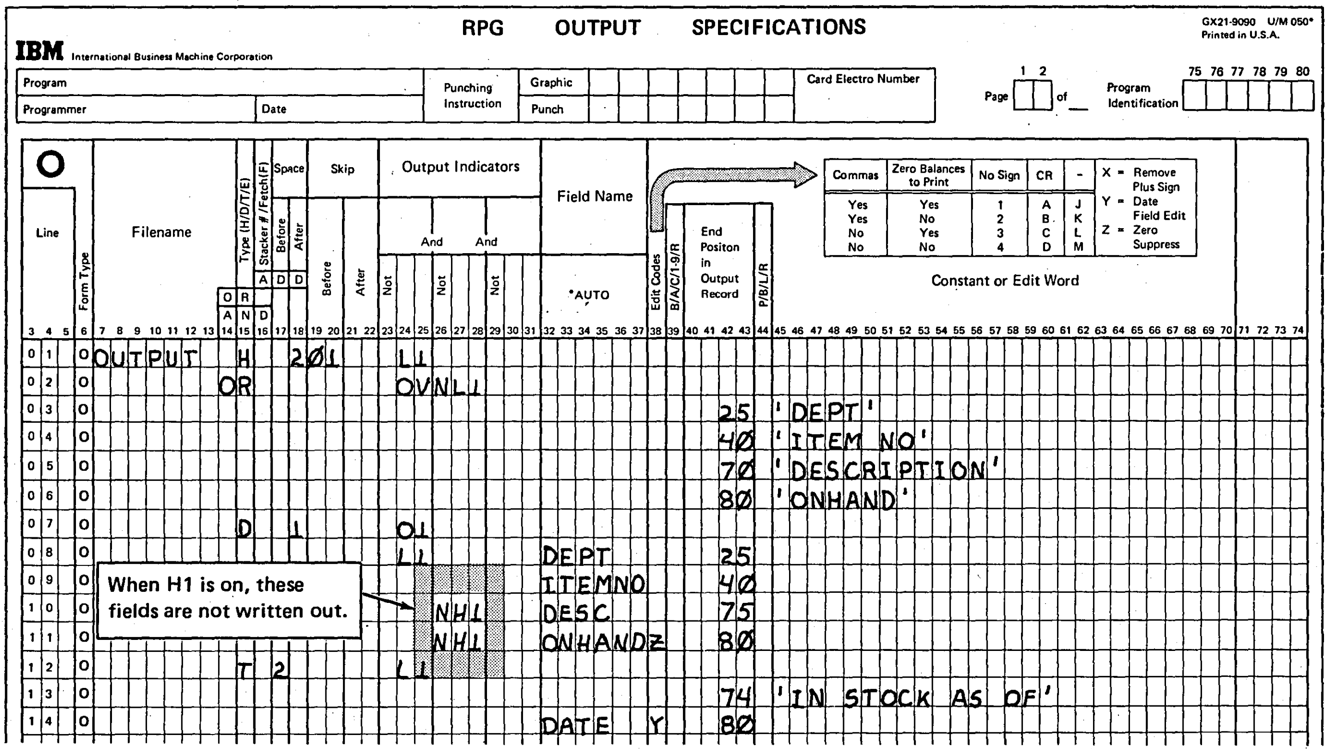
Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

```
01 OOUTPUT  H  2Ø1   L1
02 O       OR        OVNL1
03 O                                   25 'DEPT'
04 O                                   4Ø 'ITEM NO'
05 O                                   7Ø 'DESCRIPTION'
06 O                                   8Ø 'ONHAND'
07 O        D  1    NH1 Ø1
08 O                 L1      DEPT      25
0                            ITEMNO    4Ø
1                            DESC      74
1                            ONHANDZ   8Ø
12 O        T 2      L1
13 O                                   74 'IN STOCK AS OF'
14 O                         DATE  Y   8Ø
15 O
```

When H1 is on, this record is not written out.

Figure 5-3. Preventing a Record from Printing

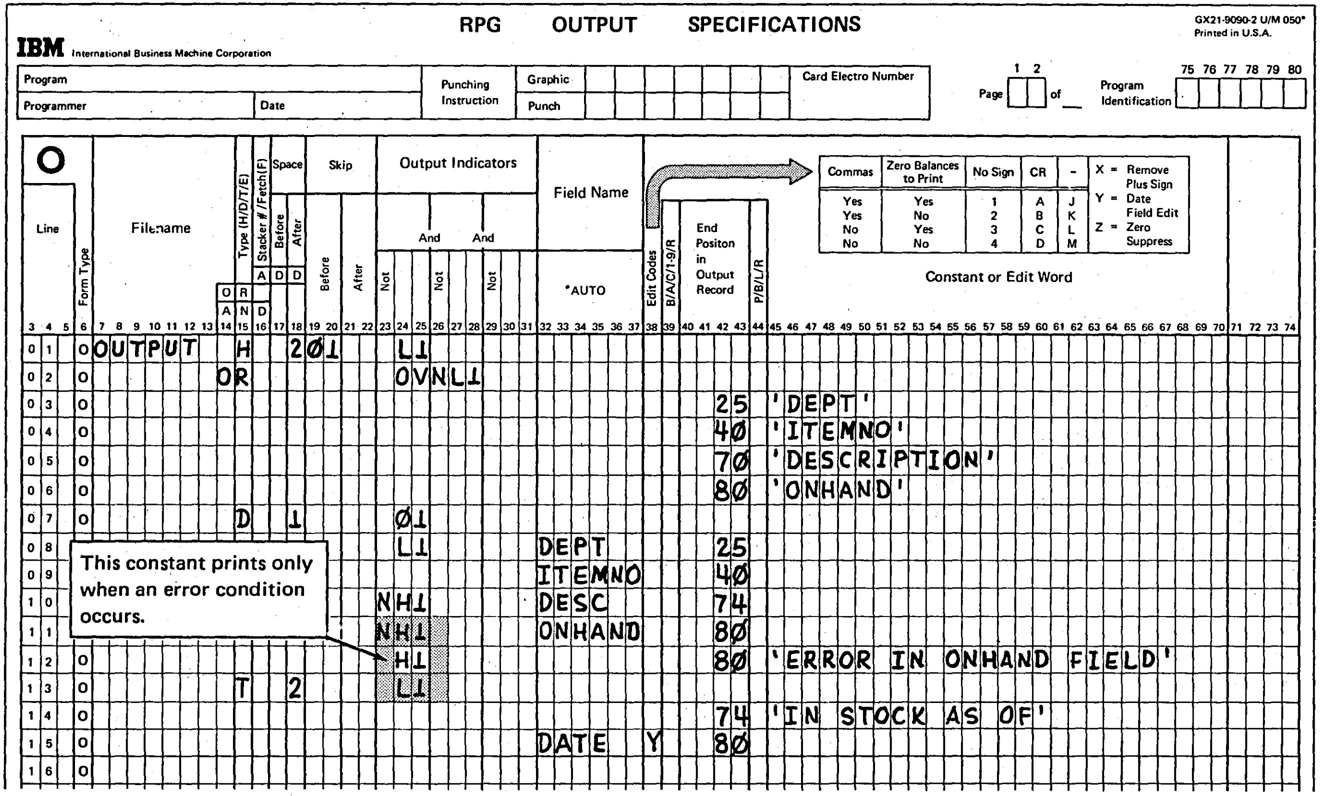


IBM International Business Machine Corporation — RPG OUTPUT SPECIFICATIONS — GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	OUTPUT	H	2	Ø1	L1				
02	O		OR			OVNL1				
03	O								25	'DEPT'
04	O								4Ø	'ITEM NO'
05	O								7Ø	'DESCRIPTION'
06	O								8Ø	'ONHAND'
07	O		D	1		O1				
08	O					L1	DEPT		25	
09	O						ITEMNO		4Ø	
10	O					NH1	DESC		75	
11	O					NH1	ONHANDZ		8Ø	
12	O		T	2		L1				
13	O								74	'IN STOCK AS OF'
14	O						DATE	Y	8Ø	

When H1 is on, these fields are not written out.

Figure 5-4. Preventing Fields from Being Written



IBM International Business Machine Corporation — RPG OUTPUT SPECIFICATIONS — GX21-9090-2 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	OUTPUT	H	2	Ø1	L1				
02	O		OR			OVNL1				
03	O								25	'DEPT'
04	O								4Ø	'ITEMNO'
05	O								7Ø	'DESCRIPTION'
06	O								8Ø	'ONHAND'
07	O		D	1		Ø1				
08	O					L1	DEPT		25	
09	O						ITEMNO		4Ø	
10	O					NH1	DESC		74	
11	O					NH1	ONHAND		8Ø	
12	O					H1			8Ø	'ERROR IN ONHAND FIELD'
13	O		T	2		L1				
14	O								74	'IN STOCK AS OF'
15	O						DATE	Y	8Ø	
16	O									

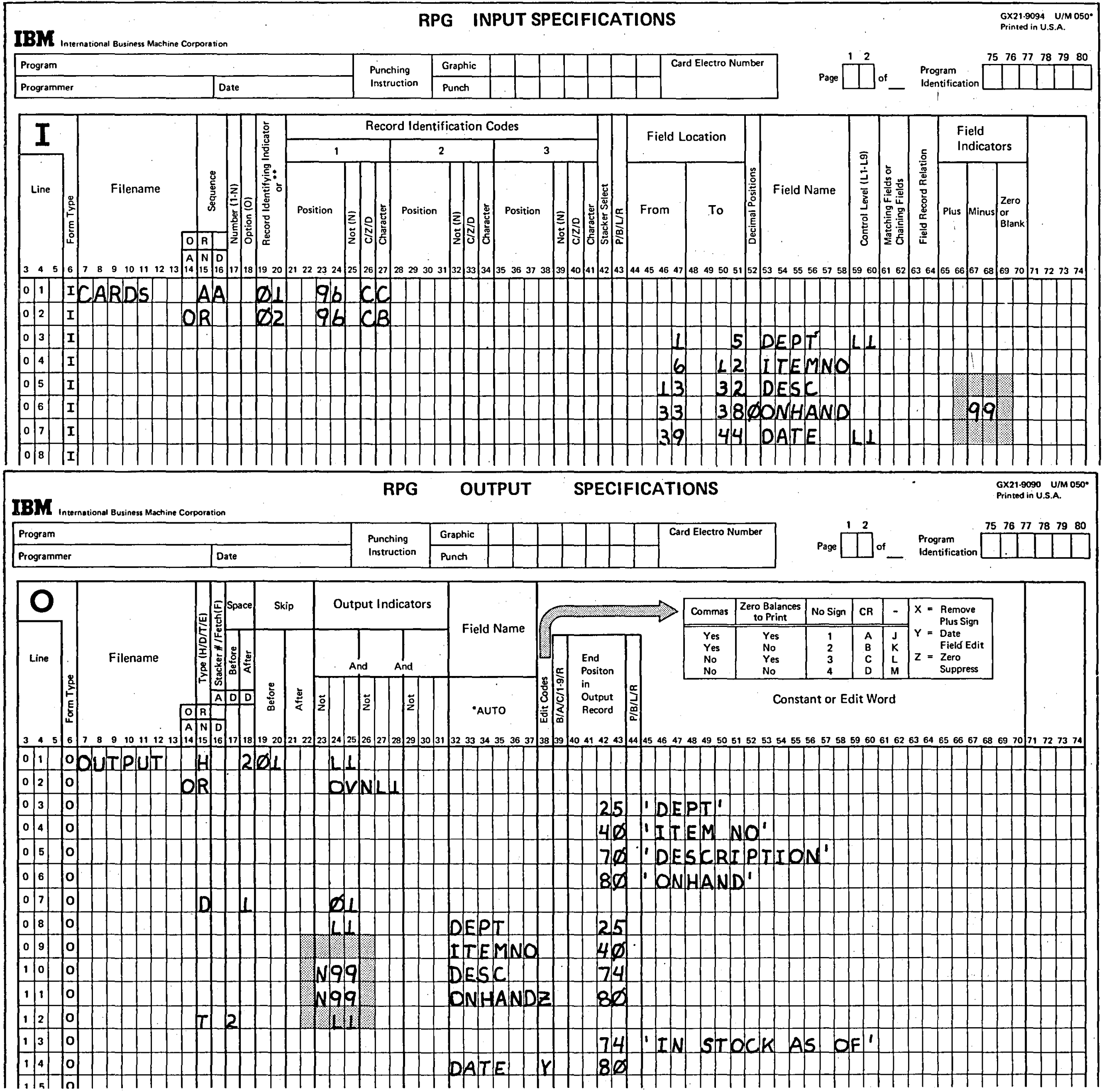
This constant prints only when an error condition occurs.

Figure 5-5. Doing Output When an Error Occurs

Using Indicators Other Than H1-H9 to Bypass Error Conditions

If you are not interested in halting when an error condition occurs but still wish to bypass the processing of erroneous data, you may use indicators 01-99 instead. They, like halt indicators, may be assigned to check for error conditions in data on the Input sheet and are then later used to condition calculations and output operations (see Figure 5-6). They do not cause a halt.

When you do not wish to halt the program for an error condition, you may select cards into a special stacker so that they will not be mixed with valid data cards. Stacker selection may be done based on the use of the indicator for an error condition.



RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

IBM International Business Machine Corporation

```
01 ICARDS   AA  01  96 CC
02 I       OR   02  96 CB
03 I                                     1   5 DEPT  L1
04 I                                     6  12 ITEMNO
05 I                                    13  32 DESC
06 I                                    33  380ONHAND        99
07 I                                    39  44 DATE  L1
08 I
```

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

IBM International Business Machine Corporation

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

```
01 OOUTPUT  H  201   L1
02 O       OR        OVNL1
03 O                                   25 'DEPT'
04 O                                   40 'ITEM NO'
05 O                                   70 'DESCRIPTION'
06 O                                   80 'ONHAND'
07 O        D  1     01
08 O                 L1      DEPT      25
09 O                         ITEMNO    40
10 O                N99      DESC      74
11 O                N99      ONHANDZ   80
12 O        T 2      L1
13 O                                   74 'IN STOCK AS OF'
14 O                         DATE  Y   80
15 O
```

Figure 5-6. Using Indicators 01-99 to Prevent Output

Controlling Which Operations are Done For a Specific Program Run

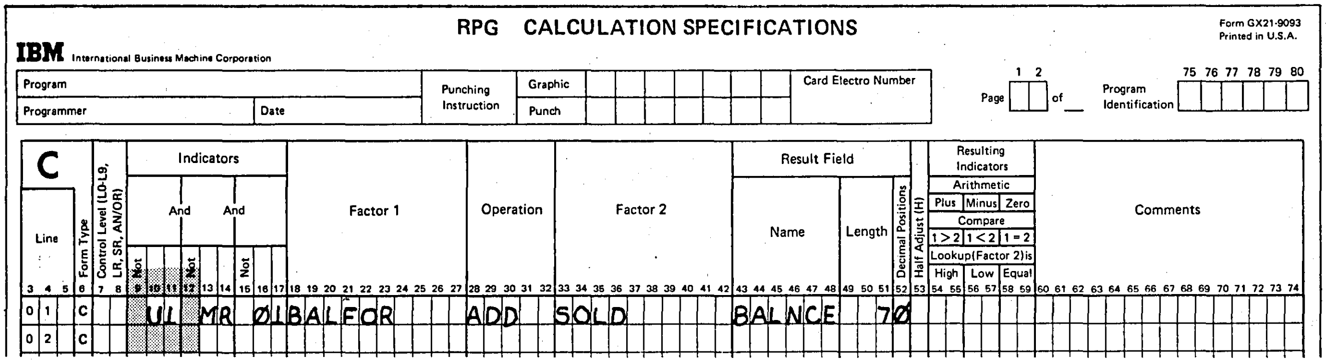
The chapter entitled *Describing and Using Input* describes how to condition the use of an input file with an external indicator so that a program can use different input files in different program runs. That chapter also describes how to assign external indicators U1-U8 on the File Description sheet and how to set the indicators.

This section describes how external indicators are used on the Calculation and Output-Format sheets to condition which operations should be done for a specific program run.

Conditioning Input Files and Related Calculations

Consider for example, calculations done for a sales analysis program. For each item in stock, monthly total sold (SOLD) is calculated and then added to last month's year-to-date total (BALFOR) to find the current year-to-date total (BALNCE). In the first month of a new year, monthly totals should not be added to prior year-to-date totals because totals are not carried over from year to year. This last statement, the year-to-date addition statement, therefore, is not done for all program runs. By conditioning the statement with an external indicator, you can control when the statement is done. In Figure 5-7, the monthly total is added to prior year-to-date only when U1 is on.

When one program is written to do two similar, yet unique, applications, some calculations may be used for both applications, some for only one. Again you may use external indicators to control which calculation specifications are used for each application.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

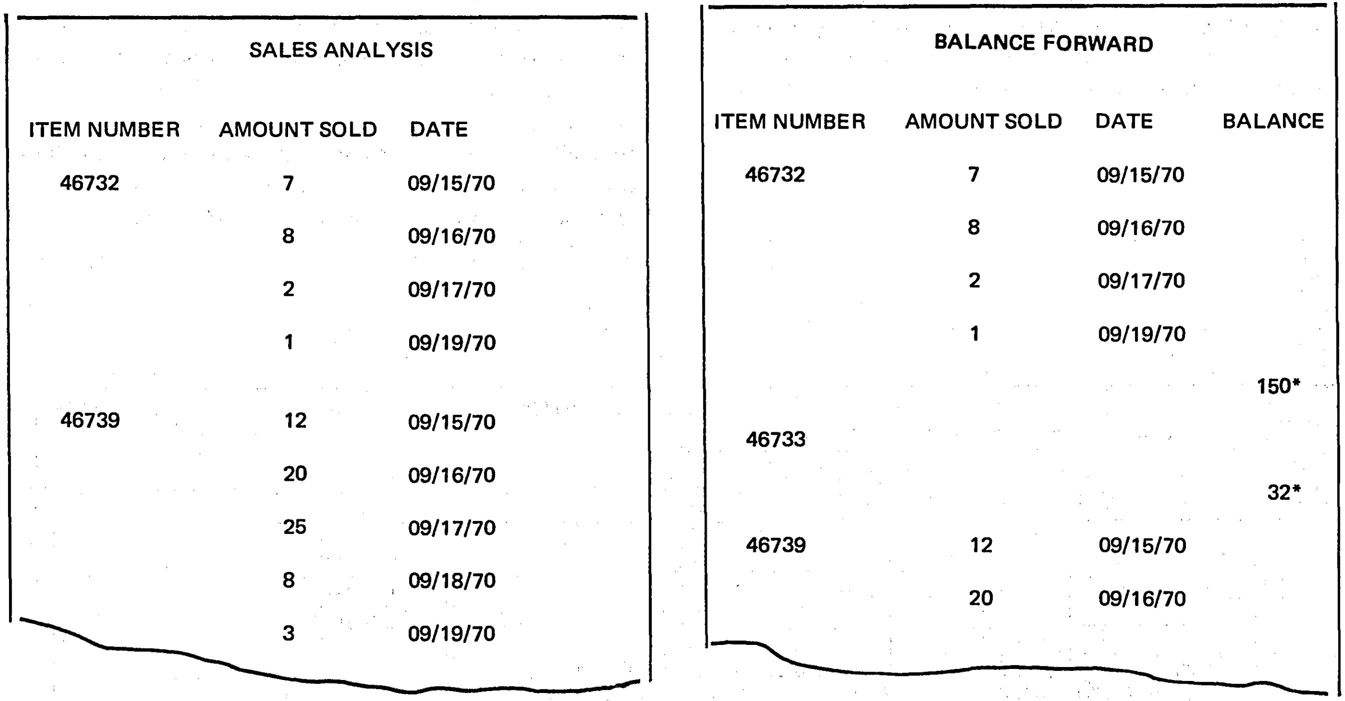
Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Result Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare 1>2 / Lookup High (54-55)	Minus / 1<2 / Low (56-57)	Zero / 1=2 / Equal (58-59)	Comments (60-74)
01	C		U1 MR Ø1	BALFOR	ADD	SOLD	BALNCE	7	Ø					
02	C													

Figure 5-7. Conditioning a Calculation by an External Indicator

Conditioning Input Files and Related Output Operations

Consider again the example discussed in the section of Chapter 2 entitled *Conditioning Use of Input Files.* Two reports were required: sales analysis and inventory (Figure 5-8). Since the results are so similar (one report merely includes more information than the other), the jobs are coded in one program.

Two files are available: a MASTER file which shows the balance forward for each item, and a TRANSACTION file which contains daily records of the number of items sold. The sales analysis job requires one file since it just creates a list of transactions. The inventory job requires two files since, for each item, it subtracts the number sold from the balance forward to find the new balance forward. An external indicator was assigned on the File Description sheet (see Figure 5-9). Its setting indicates to the program which files are to be used.



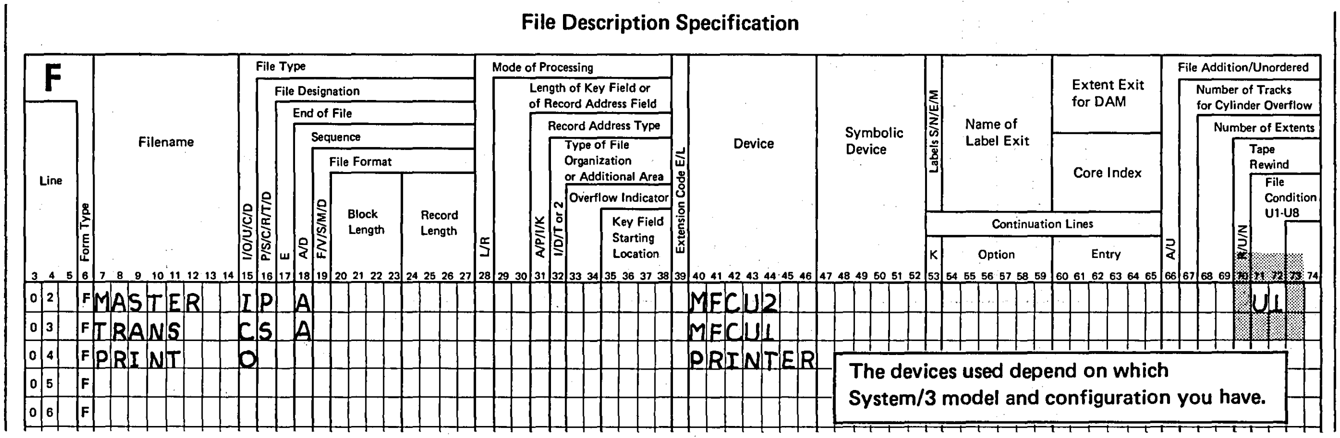
SALES ANALYSIS

ITEM NUMBER	AMOUNT SOLD	DATE
46732	7	09/15/70
	8	09/16/70
	2	09/17/70
	1	09/19/70
46739	12	09/15/70
	20	09/16/70
	25	09/17/70
	8	09/18/70
	3	09/19/70

BALANCE FORWARD

ITEM NUMBER	AMOUNT SOLD	DATE	BALANCE
46732	7	09/15/70	
	8	09/16/70	
	2	09/17/70	
	1	09/19/70	
			150*
46733			
			32*
46739	12	09/15/70	
	20	09/16/70	

Figure 5-8. Two Similar Reports



File Description Specification

Line	Form Type	Filename	File Type I/O/U/C/D	File Designation P/S/C/R/T/D	End of File E	Sequence A/D	File Format F/V/S/M/D	Block Length	Record Length	Device	Symbolic Device	File Condition U1-U8
02	F	MASTER	I	P		A				MFCU2		U1
03	F	TRANS	C	S		A				MFCU1		
04	F	PRINT	O							PRINTER		
05	F											
06	F											

The devices used depend on which System/3 model and configuration you have.

Figure 5-9. Assigning an External Indicator to the Master File

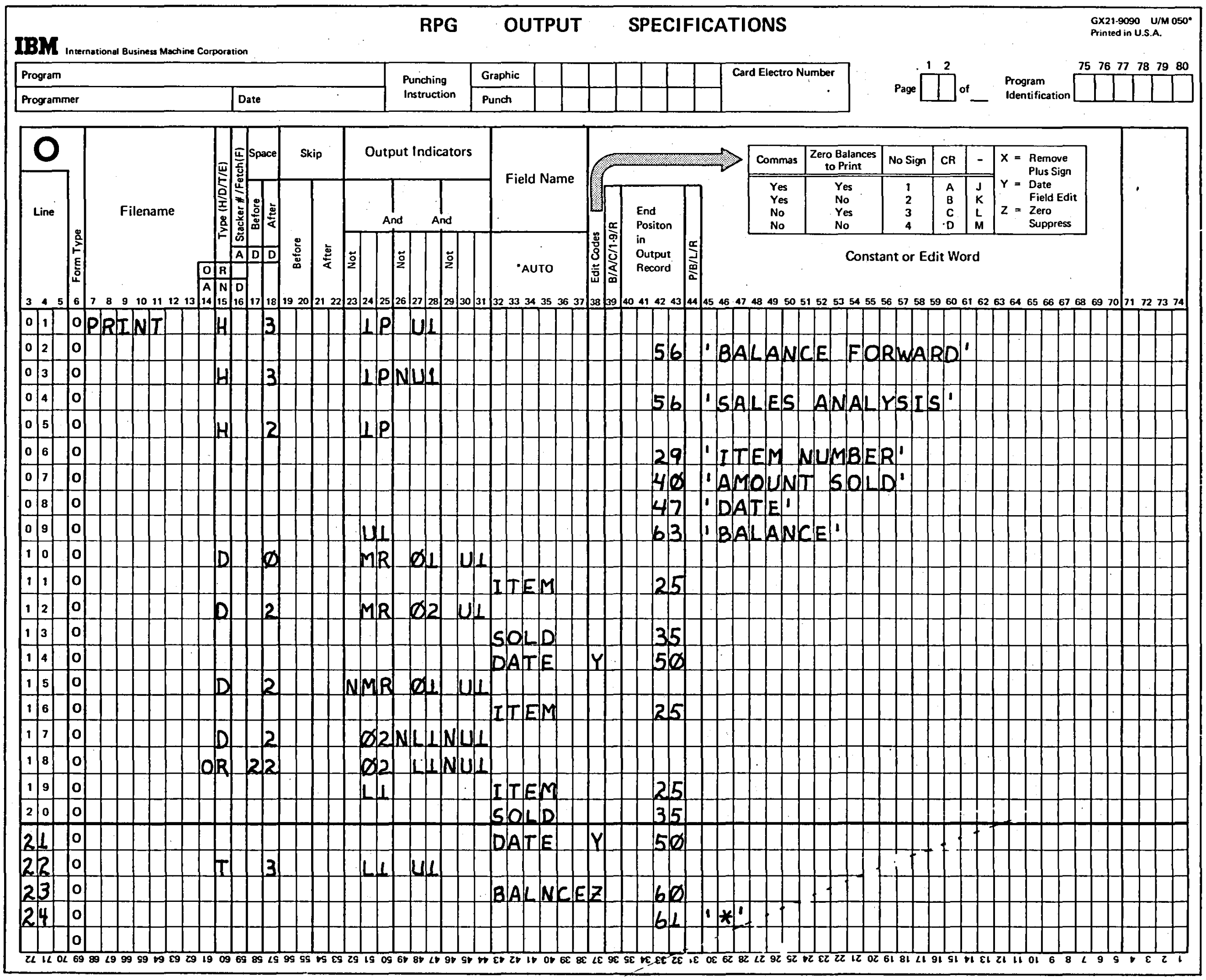
Because the results differ slightly for each job, different output operations are required. When two jobs are coded together, you indicate which operations are to be done for each through the use of an external indicator by setting the indicator to signal which files are to be used. You can specify which output operations should be done in the same way—by conditioning them by the same external indicator.

The Output-Format sheet shown in Figure 5-10 shows specifications for both jobs. Appropriate heading and detail lines are given for each. The total record is only for the balance forward job. Unless told otherwise, the computer will try to perform all specifications (provided conditions set by indicators in columns 23-31 are satisfied) in each cycle. You, therefore, have to tell the computer which operations to do for each job.

The file description specifications (Figure 5-9) show that when U1 is on, the MASTER file is used. This means that the inventory job is being done. Thus when U1 is on, only the output specifications to print records for the balance inventory are needed. Condition those output records on U1 (Figure 5-10, lines 01, 05, 10, 12, 15, 22). Condition those for the sales analysis job on NU1 (when U1 is not on the MASTER file is not used).

Conditioning Output Files and Related Output Operations

The program just discussed involves the use of a variable number of input files. One program may also require the use of a variable number of output files. In that case, the output file must be assigned an external indicator. When the indicator is on, the file is used. When it is off, the file is not used.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

```
01 OPRINT   H  3     1P U1
02 O                                   56 'BALANCE FORWARD'
03 O        H  3     1PNU1
04 O                                   56 'SALES ANALYSIS'
05 O        H  2     1P
06 O                                   29 'ITEM NUMBER'
07 O                                   4Ø 'AMOUNT SOLD'
08 O                                   47 'DATE'
09 O                 U1                63 'BALANCE'
10 O        D  Ø     MR Ø1 U1
11 O                         ITEM      25
12 O        D  2     MR Ø2 U1
13 O                         SOLD      35
14 O                         DATE  Y   5Ø
15 O        D  2    NMR Ø1 U1
16 O                         ITEM      25
17 O        D  2     Ø2NL1NU1
18 O       OR 22     Ø2 L1NU1
19 O                 L1      ITEM      25
20 O                         SOLD      35
21 O                         DATE  Y   5Ø
22 O        T  3     L1 U1
23 O                         BALNCEZ   6Ø
24 O                                   61 '*'
   O
```

Figure 5-10. Conditioning Output Operations by an External Indicator

For example, consider a sales analysis job which does the following:

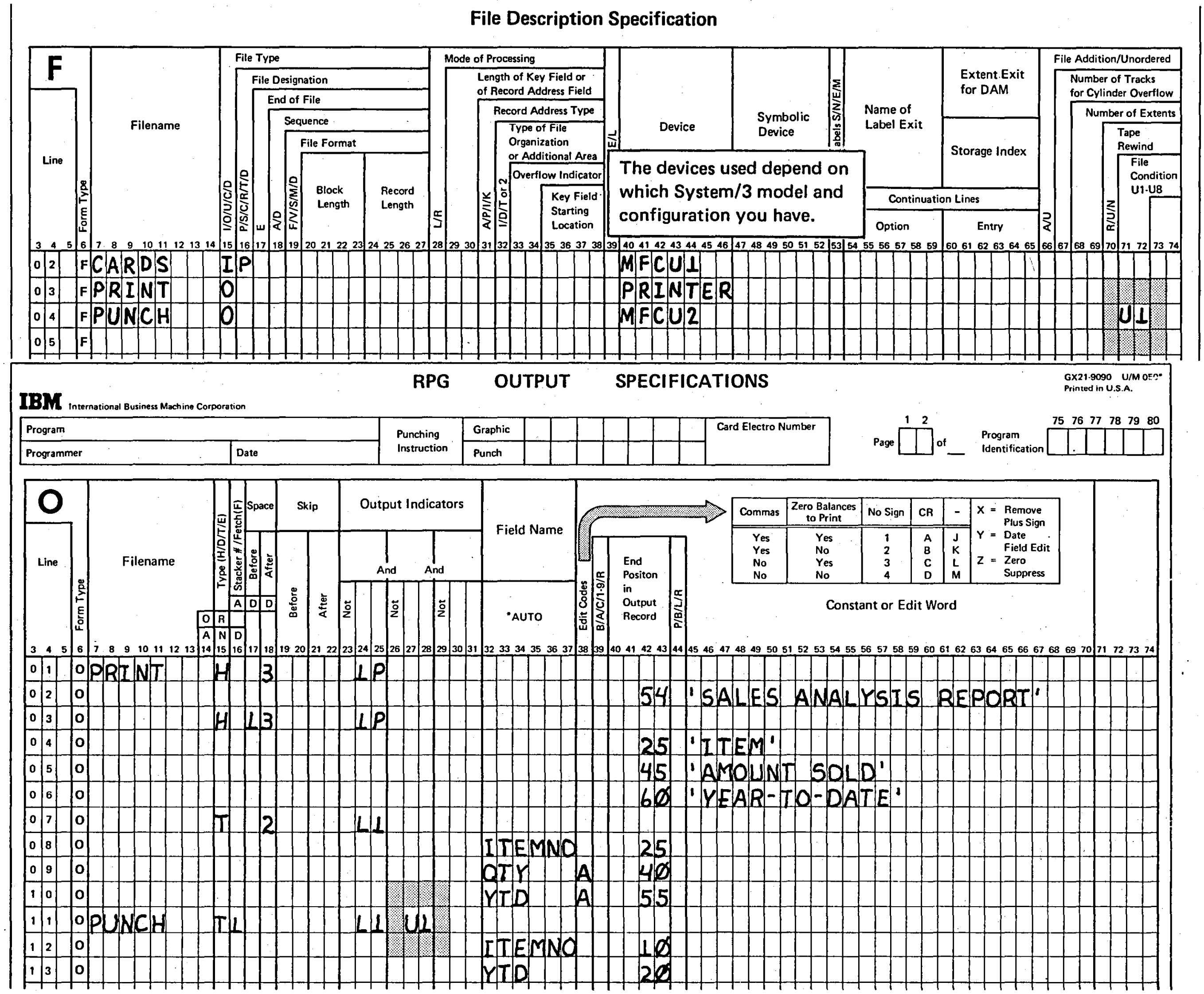
1. Calculates and records the quantity of each item sold during the month.

2. Updates the year-to-date total of the number of each item sold.

3. Creates a new year-to-date record.

The input file, organized in ascending order by item number, consists of two record types: (1) item cards, and (2) summary YTD cards. Each item card represents an item transaction. During the job, item cards are counted; and, when a control break occurs, amount sold is added to the year-to-date total found on the summary card. The number sold and current year-to-date totals are recorded on the sales analysis report, and a new summary card containing the current-year-to-date total is punched. Notice that the new year-to-date summary card is stacker-selected into stacker 1, the default stacker for the primary MFCU hopper (Figure 5-11, line 11 of the Output sheet). Assume that the old summary card is selected into a different stacker by means of an entry in column 42 of the input specifications. Thus, the new summary card is automatically placed into the item file in preparation for the next run of the program, while the old summary card can be easily discarded.

At the end of the year, new year-to-date summary cards should not be punched because the year-to-date total is not carried over into the next year. In this case, the punching operations should not be done. You can tell the program whether or not to punch by conditioning the output operations and the output file by the same external indicator. Figure 5-11 shows some of the specifications for the job.



File Description Specification

The devices used depend on which System/3 model and configuration you have.

Line	Form Type	Filename	File Type (15)	File Designation (16)	Device	File Condition U1-U8
02	F	CARDS	I	P	MFCU1	
03	F	PRINT	O		PRINTER	
04	F	PUNCH	O		MFCU2	U1
05	F					

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page ☐☐ of ___ | Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker #	Space Before	Space After	Output Indicators	And	And	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	PRINT	H			3	LP						
02	O											54	'SALES ANALYSIS REPORT'
03	O		H		1	3	LP						
04	O											25	'ITEM'
05	O											45	'AMOUNT SOLD'
06	O											60	'YEAR-TO-DATE'
07	O		T			2	L1						
08	O									ITEMNO		25	
09	O									QTY	A	40	
10	O									YTD	A	55	
11	O	PUNCH	T	1			L1	U1					
12	O									ITEMNO		10	
13	O									YTD		20	

Figure 5-11. Controlling Use of a File

Always be sure to control which calculation and output operations should be done for the job being run by conditioning the operations with the same external indicators that were assigned to the file. When you condition input files by external indicators, you *may* condition output operations if you desire. However when you condition output files, you must use the same external indicator used for conditioning the output operations. If you forget to condition the records for an output file conditioned by an external indicator, an error will occur.

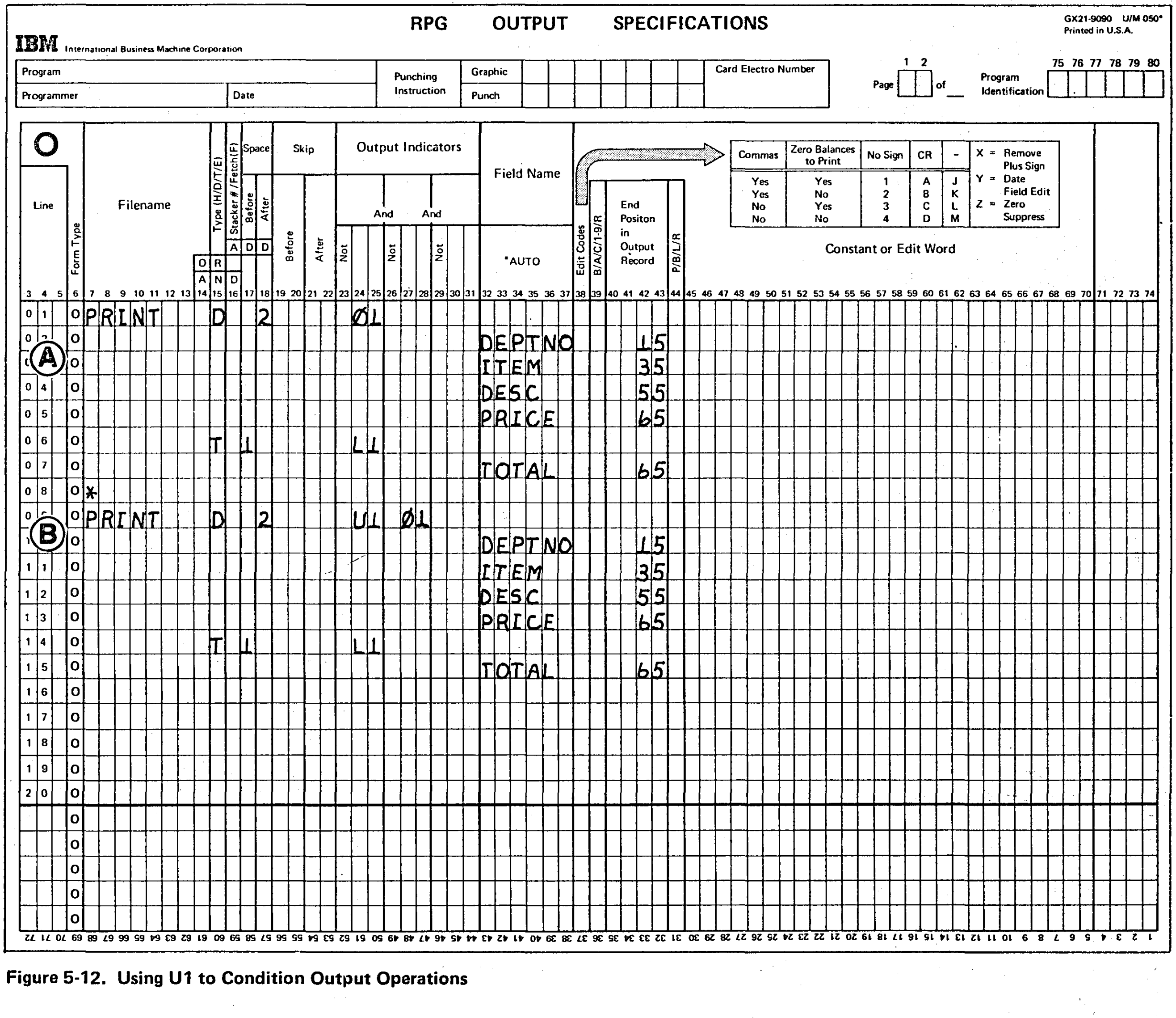
Conditioning Output Operations Only

So far, you have learned that external indicators condition files and operations related to the use of that file. External indicators need not always condition a *file*; they can condition output operations only. This means that every time the program is run, the same files will be used, but different output operations are done depending upon the setting of the external indicators.

For example, just one specification conditioned by an external indicator can change a group-printed report to a group-indicated report or vice versa. Figure 5-12, part A, shows that a detail line will print for every card. By adding an external indicator, you can control whether or not the detail line will print (Figure 5-12, part B). When U1 is on, the line prints; when U1 is not on, it will not print.

Controlling Calculations When Overflow Occurs

You normally think of using an overflow indicator to condition total or heading records that must be printed on every page of a report. But you may also use the overflow indicator to condition calculation operations. Any calculation operation so conditioned will be performed only when overflow occurs.


IBM International Business Machine Corporation
RPG OUTPUT SPECIFICATIONS
GX21-9090 U/M 050*
Printed in U.S.A.
A
B

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Output Indicators	Field Name	End Position in Output Record
01	O	PRINT	D		2	Ø1		
02	O						DEPTNO	15
03	O						ITEM	35
04	O						DESC	55
05	O						PRICE	65
06	O		T	1		L1		
07	O						TOTAL	65
08	O	*						
09	O	PRINT	D		2	U1 Ø1		
10	O						DEPTNO	15
11	O						ITEM	35
12	O						DESC	55
13	O						PRICE	65
14	O		T	1		L1		
15	O						TOTAL	65
16	O							
17	O							
18	O							
19	O							
20	O							

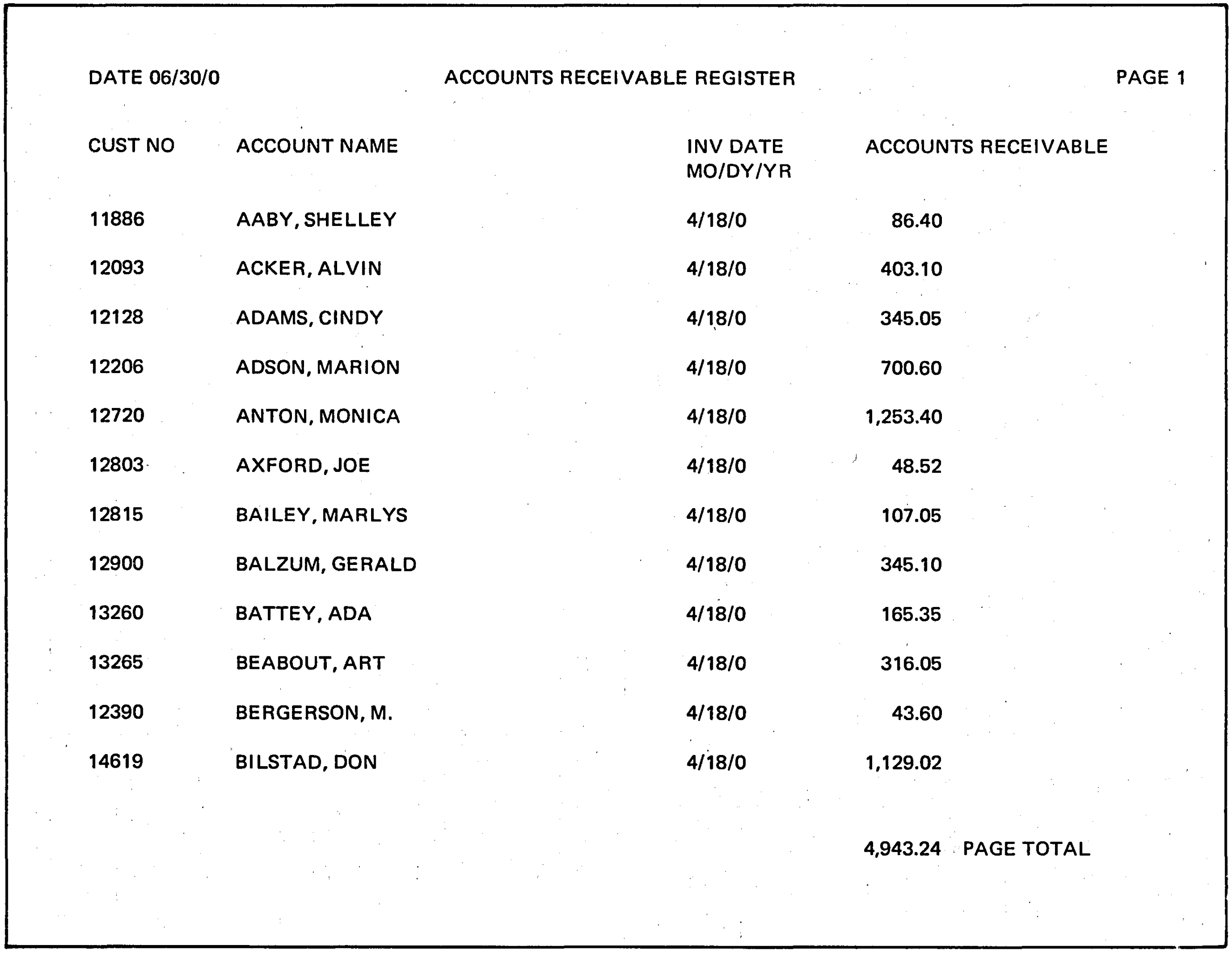
Figure 5-12. Using U1 to Condition Output Operations

Assume, for example, that you are preparing an accounts receivable report as shown in Figure 5-13. On each page of the report, you wish to have a total showing the amount of all accounts receivable on that page. You also wish to find the total amount of all accounts receivable on all pages. Thus at the beginning of each page, you must start accumulating totals for that page. When *overflow* occurs you want to add the amount of the accounts received (page total) to final total, print the page total and then reset the page total to zero so that you can start accumulating totals for the next page. Only the calculations which are to be done when overflow occurs are conditioned by the overflow indicator. See Figure 5-14 for the calculation specifications.

Performing Calculations on the Basis of the Results of Other Calculations

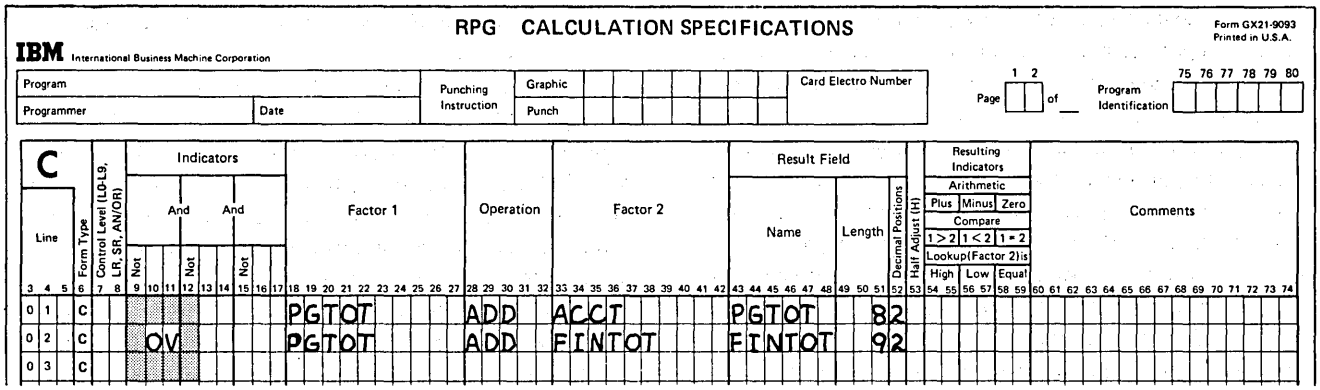
The value of the contents of a field rather than the occurrence of a certain condition can be used to determine whether or not an operation will be performed. You have worked with such situations already. For example, you have used a field on an input record to determine if processing should be done. If the field was positive, you wanted to do all calculations; if it was negative, you did *no* calculations. (See *Preventing Calculations When an Error Occurs.*)

For the situation just stated the contents of an input field determined what calculations (if any) were done. In this section, however, emphasis will be placed upon how results obtained in a calculation operation can be used to determine whether or not other calculations are performed.



```
DATE 06/30/0               ACCOUNTS RECEIVABLE REGISTER                      PAGE 1

CUST NO     ACCOUNT NAME                     INV DATE    ACCOUNTS RECEIVABLE
                                             MO/DY/YR

11886       AABY, SHELLEY                    4/18/0         86.40

12093       ACKER, ALVIN                     4/18/0        403.10

12128       ADAMS, CINDY                     4/18/0        345.05

12206       ADSON, MARION                    4/18/0        700.60

12720       ANTON, MONICA                    4/18/0      1,253.40

12803       AXFORD, JOE                      4/18/0         48.52

12815       BAILEY, MARLYS                   4/18/0        107.05

12900       BALZUM, GERALD                   4/18/0        345.10

13260       BATTEY, ADA                      4/18/0        165.35

13265       BEABOUT, ART                     4/18/0        316.05

12390       BERGERSON, M.                    4/18/0         43.60

14619       BILSTAD, DON                     4/18/0      1,129.02

                                                         4,943.24  PAGE TOTAL
```

Figure 5-13. Report with Page Totals



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus	Minus	Zero	Comments
01	C			PGTOT	ADD	ACCT	PGTOT	8	2					
02	C		OV	PGTOT	ADD	FINTOT	FINTOT	9	2					
03	C													

Figure 5-14. Conditioning a Calculation by an Overflow Indicator

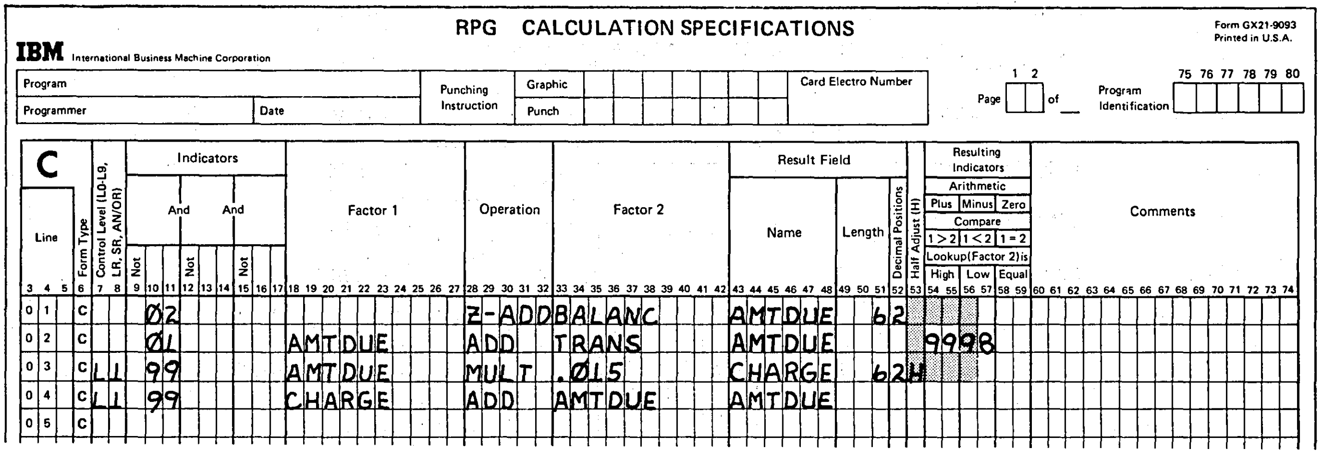
Using the Results of Arithmetic Operations

Consider how the result of a calculation can be used to determine the need for further calculations in a billing program. For each account, it is necessary to first determine the amount owed by adding charges and payments (payments are recorded as negative numbers) to the balance due at the beginning of the month. For any customer owing money at the end of the month, a service charge of 1-1/2 percent is added to the amount due. If he has credit coming, he must be sent a credit memo instead of a bill. Thus a test must be made on the amount due field to determine if it is plus or minus. If it is plus, the customer owes money and the service charge must be figured before the bill is printed. If it is minus, he has a credit and must be sent a credit memo. The card for the customer with a minus balance is stacked into a special hopper. It is later used in a credit memo run.

How can you cause a test to be made on the data? Remember in Figure 5-1 how you tested for a minus quantity. By entering an indicator (01-99, H1-H9) in columns 54-59, you can test for plus, minus, or zero depending upon where you place the indicator.

For this program, indicator 99 is placed in columns 54-55 to test for a plus condition (see Figure 5-15). When a control break occurs (all transactions for one account are processed) and when 99 is on, the 1-1/2 percent service charge is found and added to amount due (AMTDUE) to find total amount due.

If indicator 99 is off (no amount due) when the control break occurs, these last two operations are not performed.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus	Minus	Zero	Comments
01	C		Ø2		Z-ADD	BALANC	AMTDUE	6	2					
02	C		Ø1	AMTDUE	ADD	TRANS	AMTDUE				99	98		
03	C	L1	99	AMTDUE	MULT	.Ø15	CHARGE	6	2	H				
04	C	L1	99	CHARGE	ADD	AMTDUE	AMTDUE							
05	C													

● **Figure 5-15. Conditioning Calculations by an Indicator Set as a Result of an Arithmetic Operation**

The result of any arithmetic operation (ADD, SUB, MULT, DIV, Z-ADD, Z-SUB, MVR) can be tested by specifying resulting indicators in columns 54-59. The resulting indicators which are set as a result of the test can condition those operations which are to be performed on the basis of the result of that test.

Using the Results of Compare Operations

In a compare operation (COMP), fields or literals of the same type (alphameric or numeric) are compared to each other to determine their relationship to each other. Indicators entered in columns 54-59 are used to indicate whether the field or literal in Factor 1 is higher than, lower than, or equal to the field or literal in Factor 2.

The results of a compare can also control which calculations should be done next. For example, when doing an inventory and reorder application, the compare operation (COMP) is used to determine if any item needs to be reordered. In the example shown in Figure 5-16, the field called MIN (minimum) contains the critical reorder point. The field ONHAND is compared to MIN. If the amount on hand is less than or equal to MIN, indicator 99 is on. The reorder quantity is calculated by subtracting amount on hand from the number in the field called MAX which contains the maximum number which should be in stock. If amount on hand is greater than MIN, no reordering need be done and this calculation is not done.

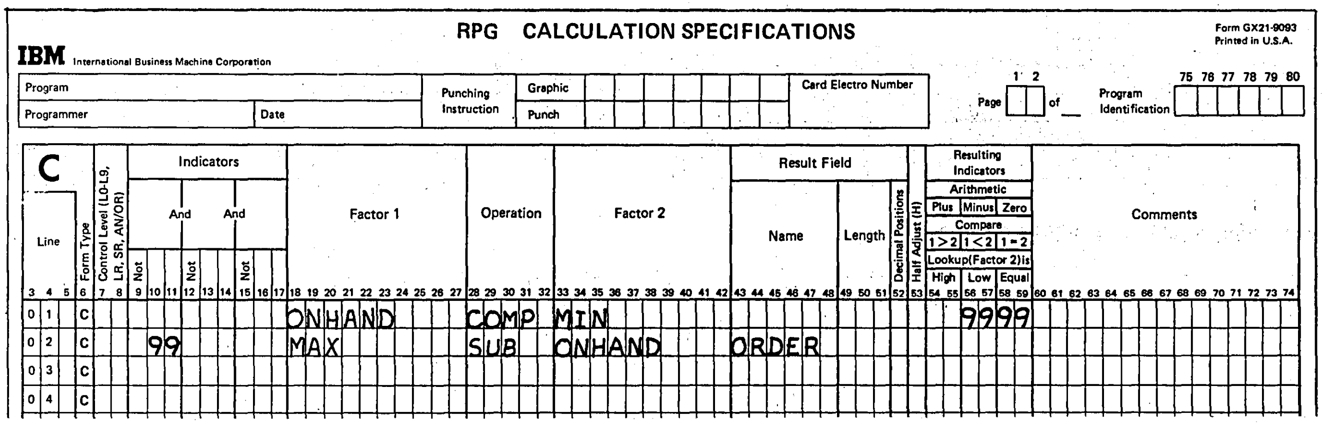
Using the Results of the Test Zone (TESTZ) Operation

Another operation code, TESTZ, is available to test data during calculations so that you can determine which calculation to do next. TESTZ tests only the *zone portion of the leftmost character of an alphameric field.* TESTZ does not test specifically for plus, minus, or zero; high, low, or equal. Rather, it tells you into which group of zones the zone tested falls:

- The zones of the character & (ampersand), A-I cause the Plus indicator entered in columns 54-55 to be turned on.
- The zones of the characters } (bracket), - (minus), and J-R cause the Minus indicator entered in columns 56-57 to be turned on.
- The zones of all other characters cause the indicator entered in columns 58-59 to be turned on.

The test zone operation could prove very useful in a large billing application. Consider the case of a company which has so many accounts that billing must be divided. Customers whose last names are in the first part of the alphabet are billed on the 15th of the month; all others are billed at the last of the month. The master file used in billing is organized in ascending order according to account number.

The records in this file could be sorted by name so that you could divide the file for billing. However, this file is used so often for other purposes that it is a waste of time to repeatedly sort it according to name and then sort it again according to account number.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

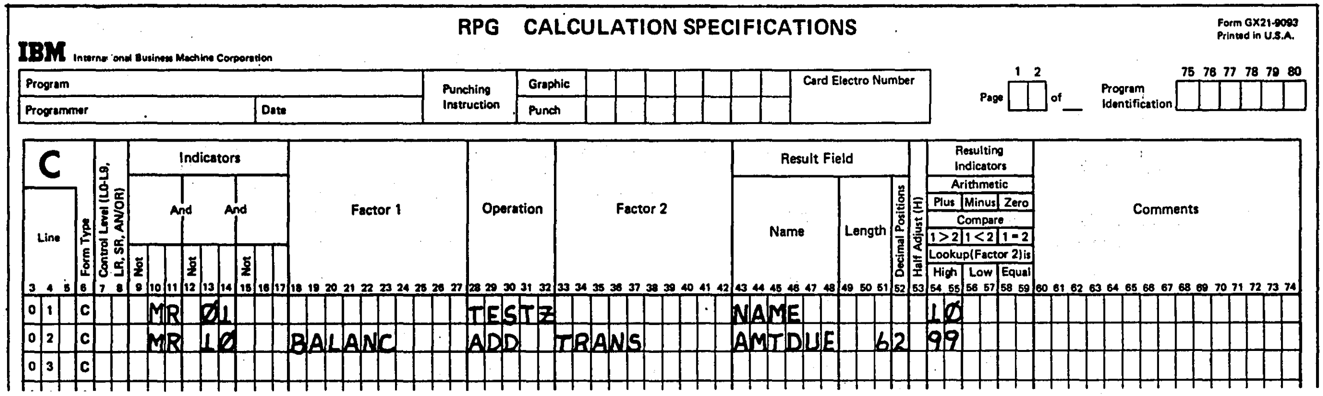
Form GX21-9093
Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Resulting Indicators High/Plus 54-55	Low/Minus 56-57	Equal/Zero 58-59	Comments
01	C		ONHAND	COMP	MIN				99	99	
02	C	99	MAX	SUB	ONHAND	ORDER					
03	C										
04	C										

Figure 5-16. Conditioning Calculations by an Indicator Set as a Result of a Compare Operation

A better way to do the billing is to test the name field in each record to see in which part of the alphabet the name falls. During the first of the month, if the last name begins with letters A-I, you wish to find amount due. TESTZ will test the first letter in the field and tell you in what part of the alphabet it is. Figure 5-17 shows the calculation specifications necessary to bill customers whose last names fill into category A-I.

Naturally at the end of the month you will want to bill the rest of the customers. But you don't want to write another program for end of the month billing. So you write one program to do both jobs and use external indicators to condition the specifications for each job (see Figure 5-18).



RPG CALCULATION SPECIFICATIONS

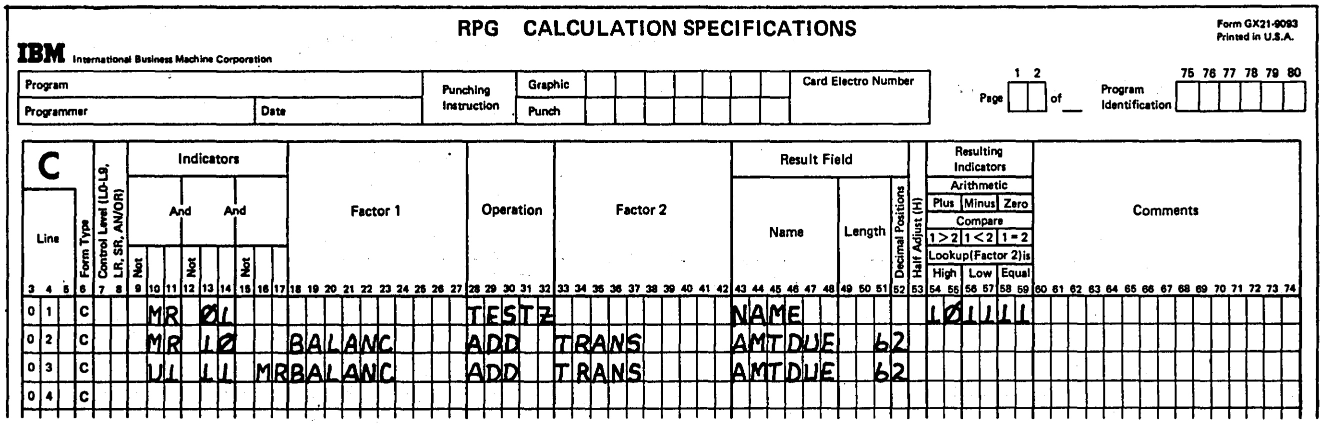
IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page of
Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Not	Indicators	Not	And	Not	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
01	C			MR		Ø1				TESTZ		NAME				LØ			
02	C			MR		LØ			BALANC	ADD	TRANS	AMTDUE	6	2		99			
03	C																		

Figure 5-17. Conditioning a Calculation by an Indicator Set as a Result of the TESTZ Operation



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page of
Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Not	Indicators	Not	And	Not	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
01	C			MR		Ø1				TESTZ		NAME				LØ	L1	L1	
02	C			MR		LØ			BALANC	ADD	TRANS	AMTDUE	6	2					
03	C			U1		L1		MR	BALANC	ADD	TRANS	AMTDUE	6	2					
04	C																		

Figure 5-18. Using TESTZ and External Indicators

You can use TESTZ to test for any special code you set up by using the zone of a character. This is most often done when you have no space on your records for any other kind of identifying information. For example, when establishing a code for the percentage of commission received by each salesman, you could use the & to indicate 6 percent and the minus (-) sign to indicate 15 percent. You would, of course, have to punch this code in the leftmost position of a numeric field because this is the position tested by the TESTZ operation. Figure 5-19 shows how the code is placed in the field containing salesman number. However, you must define the field as alphameric since the TESTZ operation can only be performed on an alphameric field.

Figure 5-20 shows the TESTZ used on the SALSNO (salesman number) field, which contains the commission code, in order to find rate of commission. The results of the test determines what other calculations will be done.

SALESMAN -

Whose number is 17657778

Who earns 15% commission

Has 17657778 punched in SALSNO field

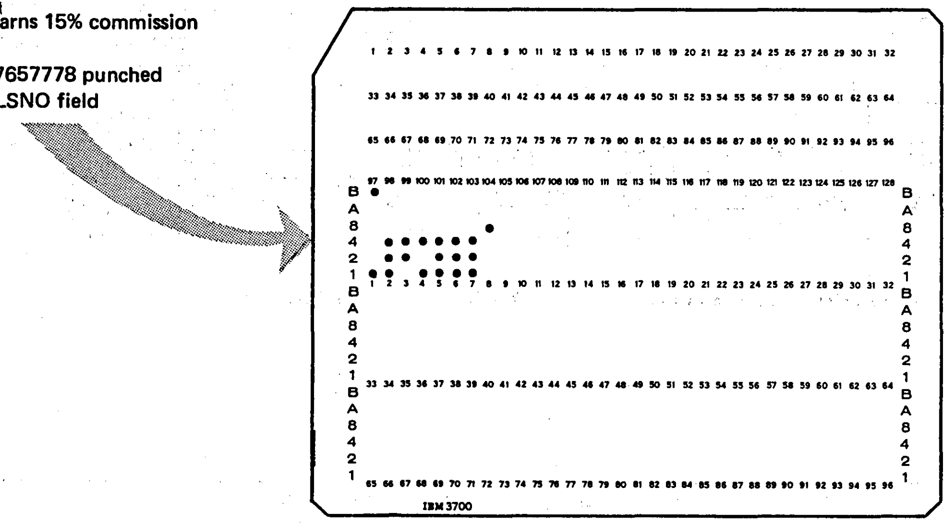
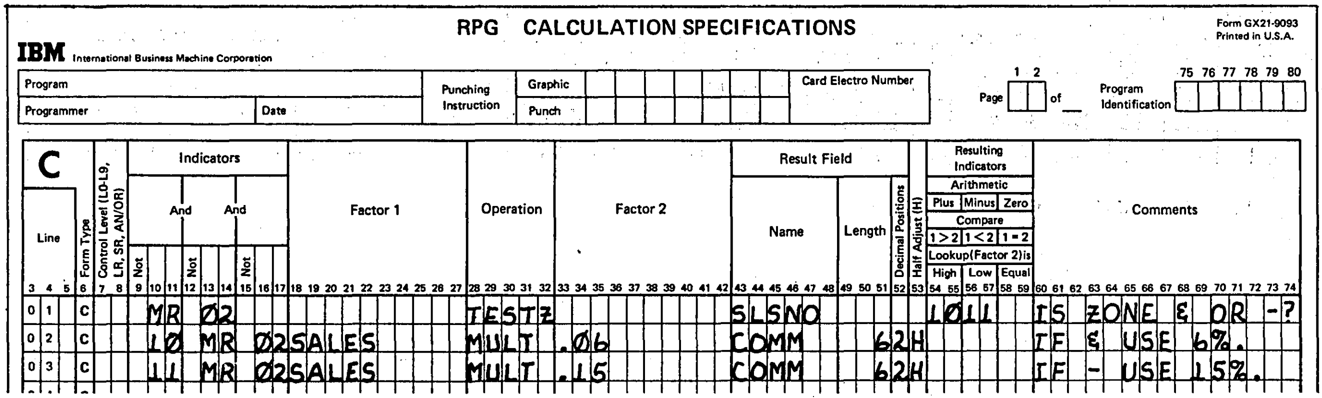


Figure 5-19. Punching a Code



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page (1 2) of
Program Identification (75 76 77 78 79 80)

C

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not (9)	Indicators (10-11)	And: Not (12)	And (13-14)	And: Not (15)	And (16-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field: Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / High / 1>2 (54-55)	Minus / Low / 1<2 (56-57)	Zero / Equal / 1=2 (58-59)	Comments (60-74)
01	C			MR		Ø2				TESTZ		SLSNO				1Ø	11		IS ZONE & OR -?
02	C			1Ø		MR		Ø2	SALES	MULT	.Ø6	COMM	6	2	H				IF & USE 6%.
03	C			11		MR		Ø2	SALES	MULT	.15	COMM	6	2	H				IF - USE 15%.

Figure 5-20. Testing a Field to Determine a Code

CONTROLLING OPERATIONS ON THE BASIS OF THE NEXT RECORD IN A FILE

Sometimes, calculations to be performed may depend upon information in the next record or on the type of the next record to be processed. For example, in a certain kind of program, you might want to bypass calculations for the current record if you know the next record in the file is identical.

The RPG II language has a special feature called *look ahead,* which extends the basic RPG II logic. It will allow the computer to look at information in the next record to be processed while it is processing the current record. This means that information in record B can be used while record A is being processed. By using this feature, you can write a program that uses information from the next record available for processing.

Look ahead can be used with card, tape, or disk input files. This section discusses look ahead with card (MFCU) and disk files. For MFCM, tape, and other files, the concept is similar.

Processing Card or Disk Files

MFCU Files: (Refer to the representation of the MFCU card path in Figure 5-21 during this discussion.) As Card A is read, data recorded on it is transferred to the input area. The card then moves on to the wait station. According to the RPG II program cycle, information is transferred from the input area to the processing area right before detail time. At detail time, then, calculations can be done on data from the card path which is in the wait station (Card A).

However, when look ahead is specified, another card (Card B) is read before detail time operations are performed in the current cycle. Card A is stacked and information from Card A is moved to the processing area. Then information on Card B just read is transferred to the input area and is available for use while processing Card A, now in the stacker.

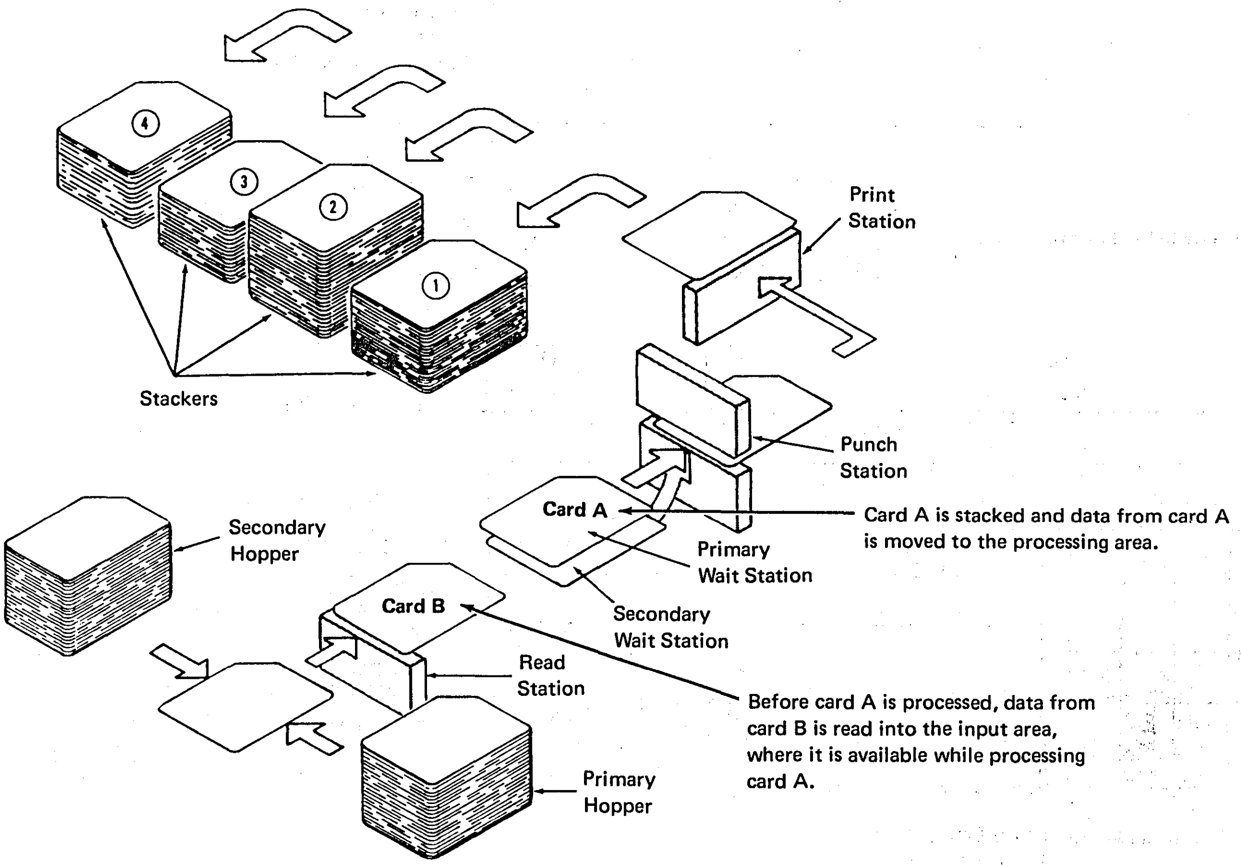
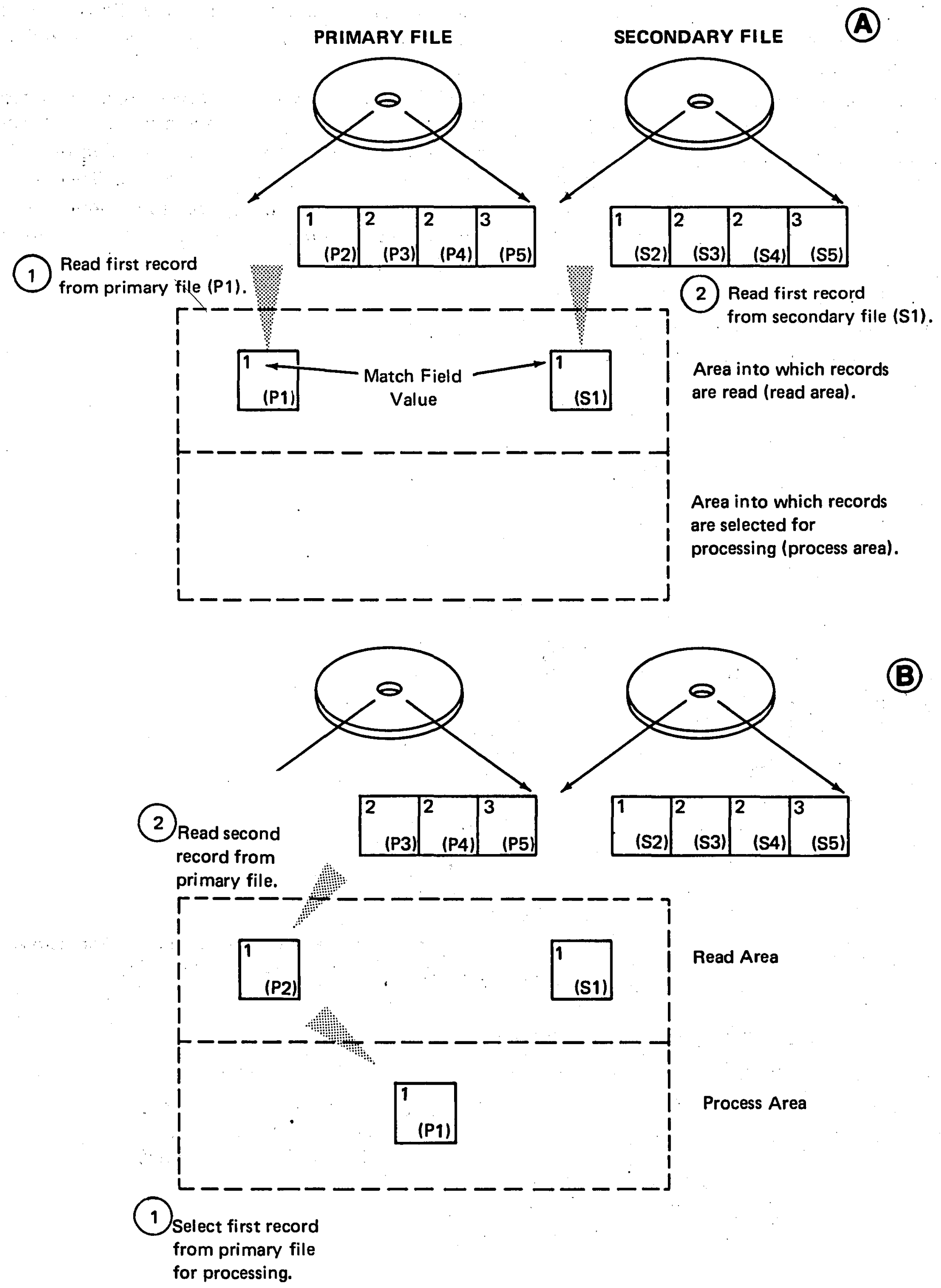

4
3
2
1
Print Station
Stackers
Punch Station
Card A
Card A is stacked and data from card A is moved to the processing area.
Secondary Hopper
Primary Wait Station
Card B
Secondary Wait Station
Read Station
Before card A is processed, data from card B is read into the input area, where it is available while processing card A.
Primary Hopper

Figure 5-21. The Look Ahead Function with a Card File

Disk Files: Figure 5-22 shows processing of three of the records from two disk input files, one primary and one secondary. The records available for look ahead during the processing of these records are:

Record Processed	*Records Available*
P1	P2 and S1
P2	P3 and S1
S1	P3 and S2



PRIMARY FILE
SECONDARY FILE
Ⓐ
1 (P2)
2 (P3)
2 (P4)
3 (P5)
1 (S2)
2 (S3)
2 (S4)
3 (S5)
① Read first record from primary file (P1).
② Read first record from secondary file (S1).
1 (P1)
Match Field Value
1 (S1)
Area into which records are read (read area).
Area into which records are selected for processing (process area).
Ⓑ
② Read second record from primary file.
2 (P3)
2 (P4)
3 (P5)
1 (S2)
2 (S3)
2 (S4)
3 (S5)
1 (P2)
1 (S1)
Read Area
1 (P1)
Process Area
① Select first record from primary file for processing.

Figure 5-22 (Part 1 of 2). Records Available for Look Ahead: Two Input Files

In general, when the record being processed is from an input file, the next record in the input file is available as are the records which were read but not processed from the other files.

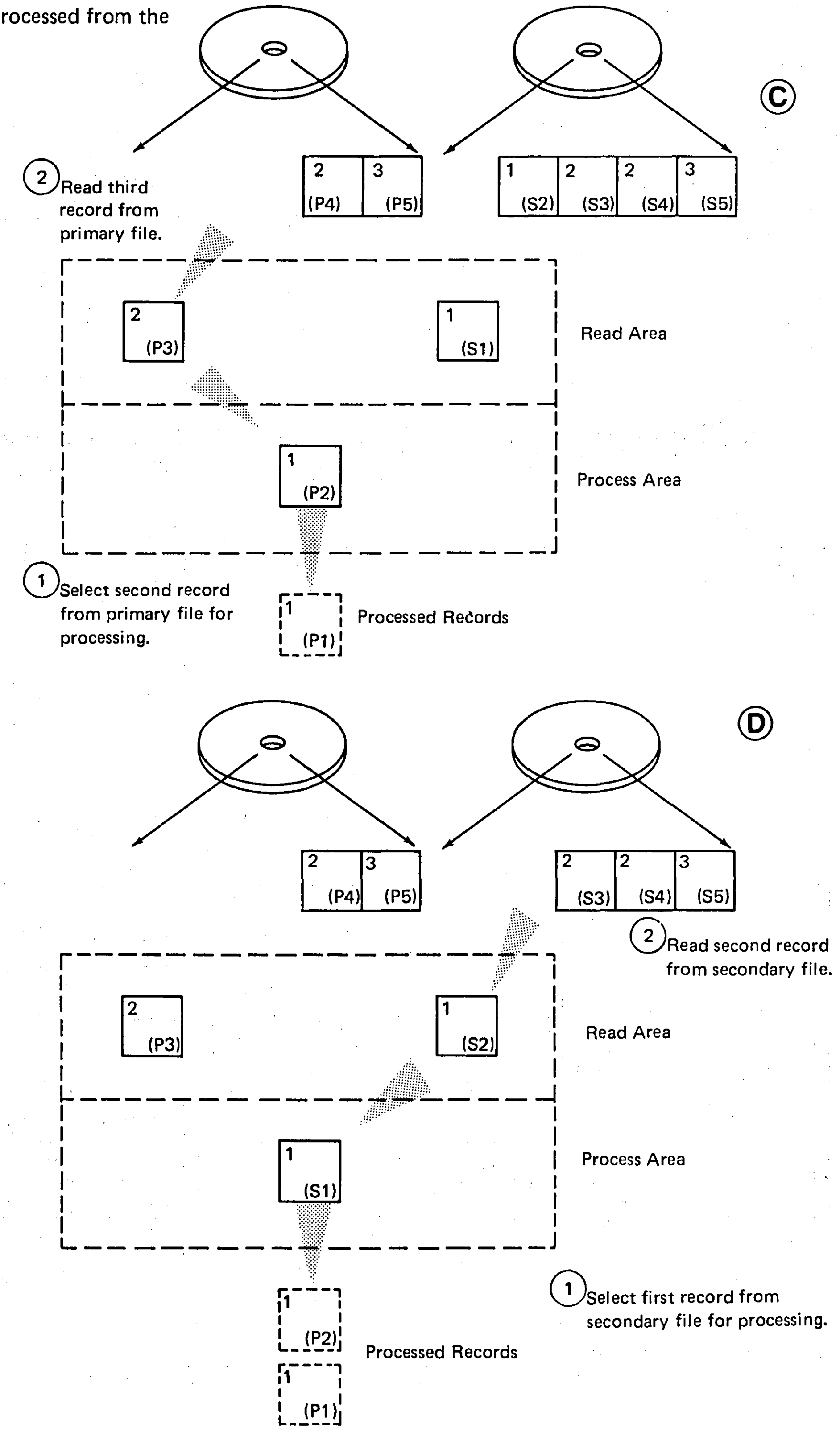


Figure 5-22 (Part 2 of 2). Records Available for Look Ahead: Two Input Files

Checking for Duplicates

Duplicate records or records with duplicate fields are sometimes considered erroneous. Only one of the duplicates should be used for the job.

Consider, for example, the case of a company which has a large turnover in inventory items. Quite frequently new items are added and others deleted from the inventory. A number for a deleted part is to be assigned to a new part. Some mistakes have occurred, however, and one part number has been assigned to two different items. As a result of this error, inventory balances for these items have not been updated correctly, and errors have been made on customer invoices. If this situation is possible, a regular check should be made for duplicate part numbers.

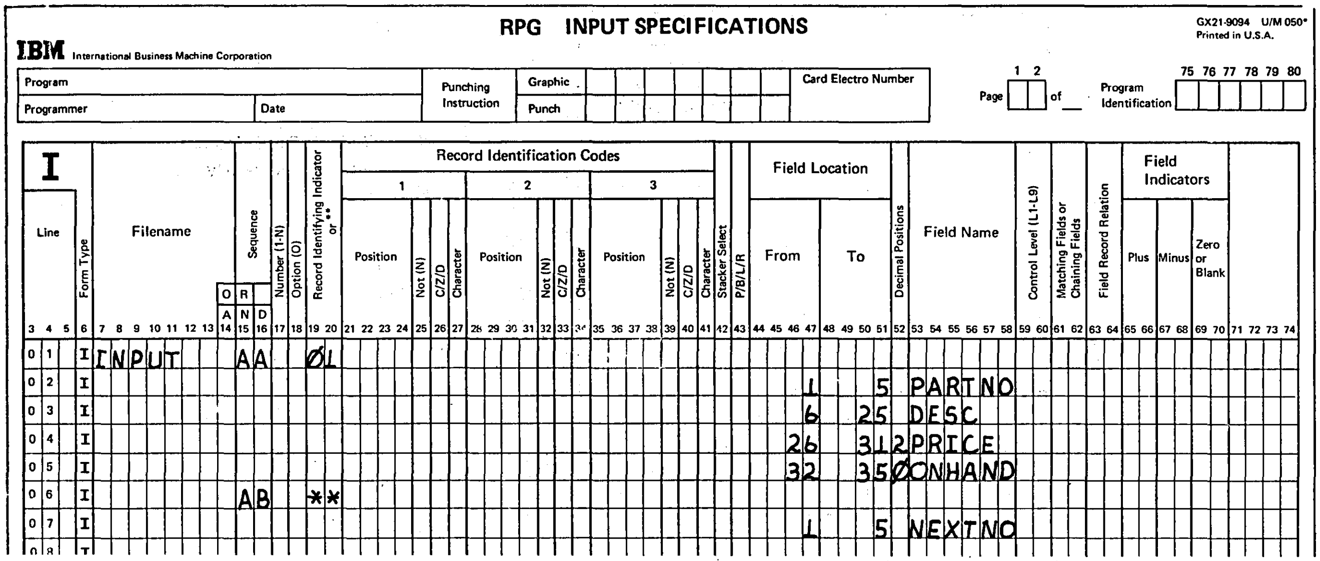
Each month, a report is created showing the complete inventory. All part numbers are listed on the report. You could look through the report to check for duplicate part numbers, but it would be easier and more accurate if you could add a few specifications that would check for duplicates and indicate on the report which item numbers are duplicate.

By using the look ahead feature you have access to information that is coming up. You can then use this information to determine what operations to do. If you are processing a record with part number 64322, and you know that the next record also has part number 64322, you can print a message indicating duplicate part numbers, then halt. But, if you are processing the record with part number 64322 and you do not know that the next record also has part number 64322, you can do nothing special because you are not aware that you are processing a record which contains a duplicate entry.

Writing Specifications for Look Ahead

Any field which you want to look at in the next record to be processed must be defined as a look ahead field. If that field is also used in normal processing (other than as a look ahead field), it must be defined in the normal way also. Thus, most look ahead fields will be specified twice.

Figure 5-23, lines 01-05, shows specifications needed to describe the input file used in preparing an inventory listing. When checking for duplicates, PARTNO is the field you want to use when looking ahead at the next record; therefore, PARTNO must be defined as a look ahead field. The specifications in Figure 5-23, lines 06-07, do this.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

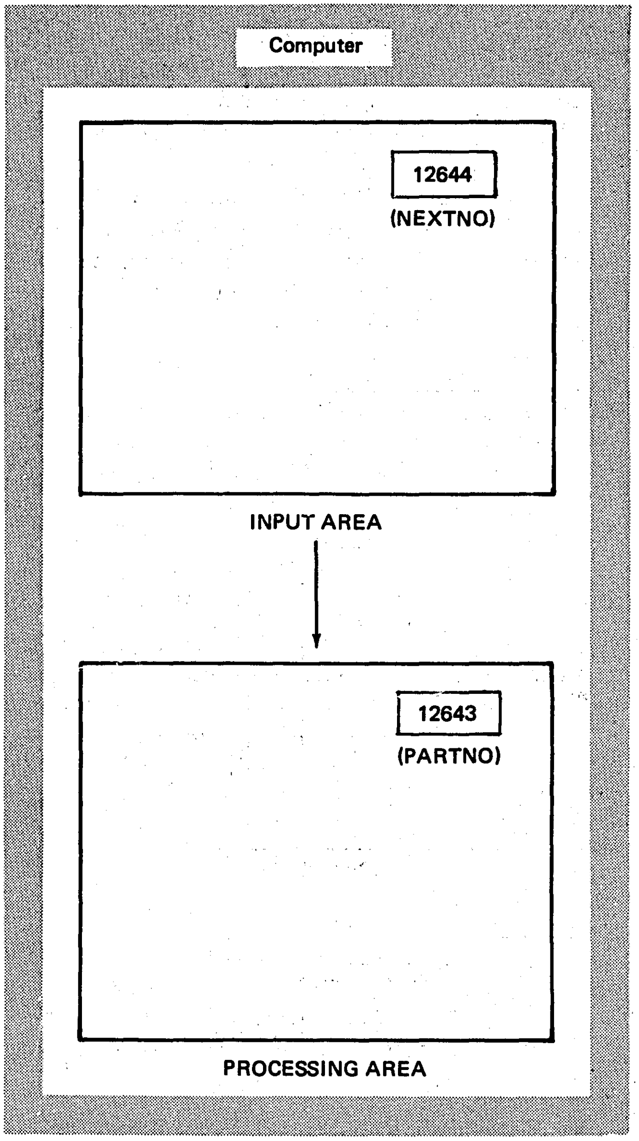
Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Filename	Sequence	Record Identifying Indicator or **	Field Location From	Field Location To	Decimal Positions	Field Name
01	I	INPUT	AA	Ø1				
02	I				1	5		PARTNO
03	I				6	25		DESC
04	I				26	31	2	PRICE
05	I				32	35	Ø	ONHAND
06	I		AB	**				
07	I				1	5		NEXTNO
08	I							

Figure 5-23. Look Ahead Specifications

All look ahead fields must be defined as being in a record type different from the others defined. This is done by using a unique alphabetic sequence entry in columns 15-16. No record identifying indicator (01-99) can be used. A double asterisk (**) is placed in columns 19-20 to specify that the fields described in the following lines are look ahead fields. Field location is also specified for look ahead fields.

Every look ahead field must be named, but the name given must be different than when it was described as a normal input field. The same field is given two names so that you can distinguish between the field on the record being processed and that same field (the look ahead field) on the record that is to be processed next (Figure 5-24).



NEXTNO refers to positions 1-5 in the record to be processed next.

PARTNO refers to positons 1-5 in the record currently processed.

Figure 5-24. Look Ahead Field: A Field with Two Names

Using Look Ahead Information

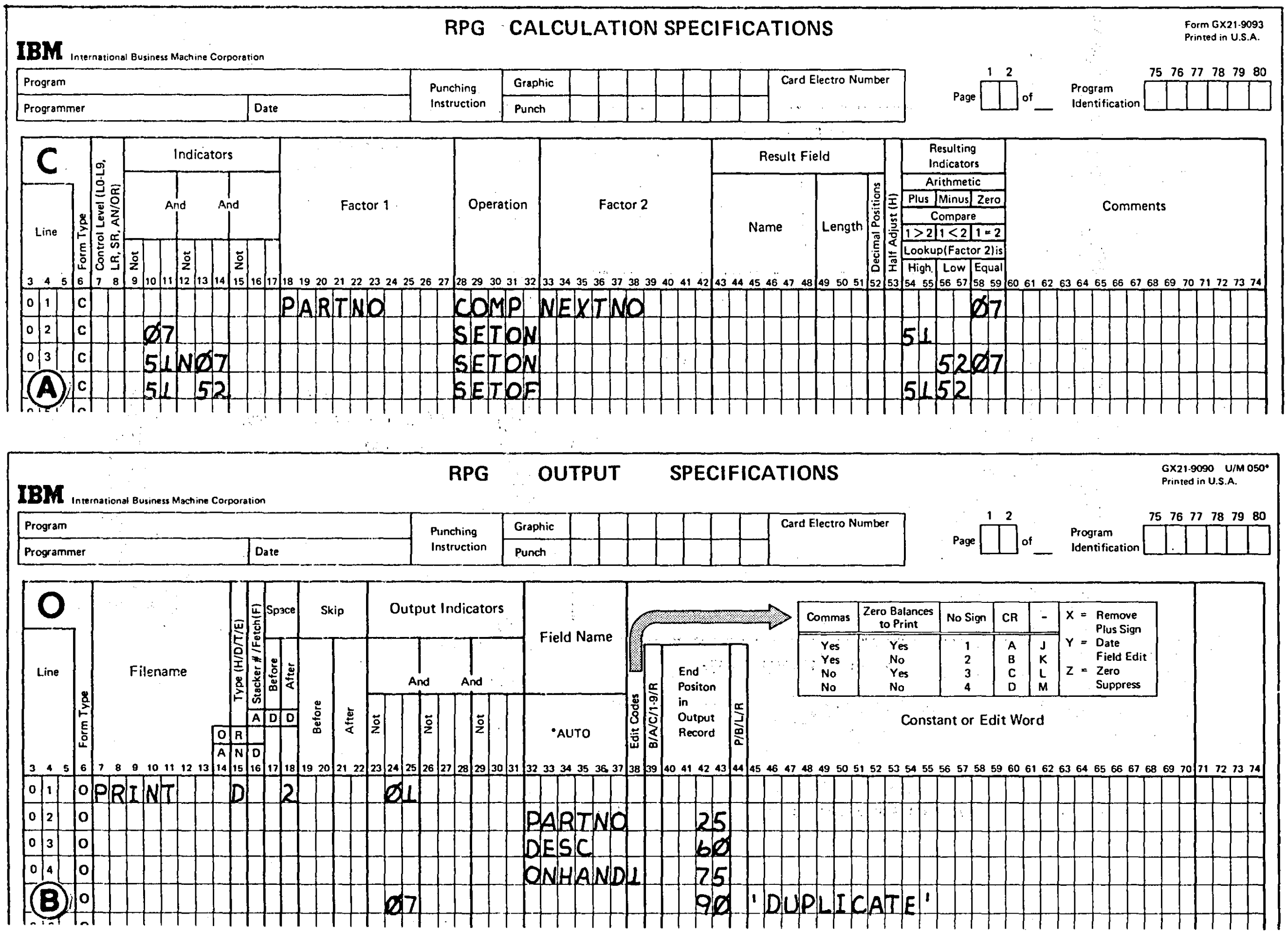
Now that you have specified the look ahead field, you can use it as you would any other field. The only exceptions are that you cannot use it as a result field in calculations, nor can it be blanked after for output.

In the listing program, you have to make a comparison between part numbers from two records. If PARTNO on the record being processing is the same as NEXTNO on the next record to be processed, you wish to print a message indicating duplicate entries. If the PARTNO and NEXTNO fields do not match, there are no duplicates for that part number, and the item is merely listed.

Figure 5-25 shows specifications for the program. The operation in line 01 of the Calculation sheet compares the part number on the record being processed (PARTNO) to the part number on the record coming next (NEXTNO). If they are equal, indicator 07 is turned on. Notice on the Output-Format sheet that when 07 is on, the word *duplicate* is printed.

The SETON and SETOF operations in lines 02-04 of the Calculation sheet are used so that the computer will indicate a duplicate when the second record having the duplicate part number is processed.

Consider, for example, records A1, A2, and B. The first two records are duplicates; the third is not. When A1 is processed, the program looks ahead to A2 and, by comparing, knows that A2 is the same as A1. When A2 is processed, the program looks ahead to B. The compare will say that A2 is not a duplicate since it does not match B1. But A2 really is a duplicate because it is the same as A1. Thus, when processing A1, you have to set an indicator which will be on when A2 is processed and which will indicate that A2 is a duplicate since it matches the previous record.



RPG CALCULATION SPECIFICATIONS

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	High	Low	Equal
01	C		PARTNO	COMP	NEXTNO			Ø7
02	C	Ø7		SETON		51		
03	C	51 NØ7		SETON			52	Ø7
Ⓐ	C	51 52		SETOF		51	52	

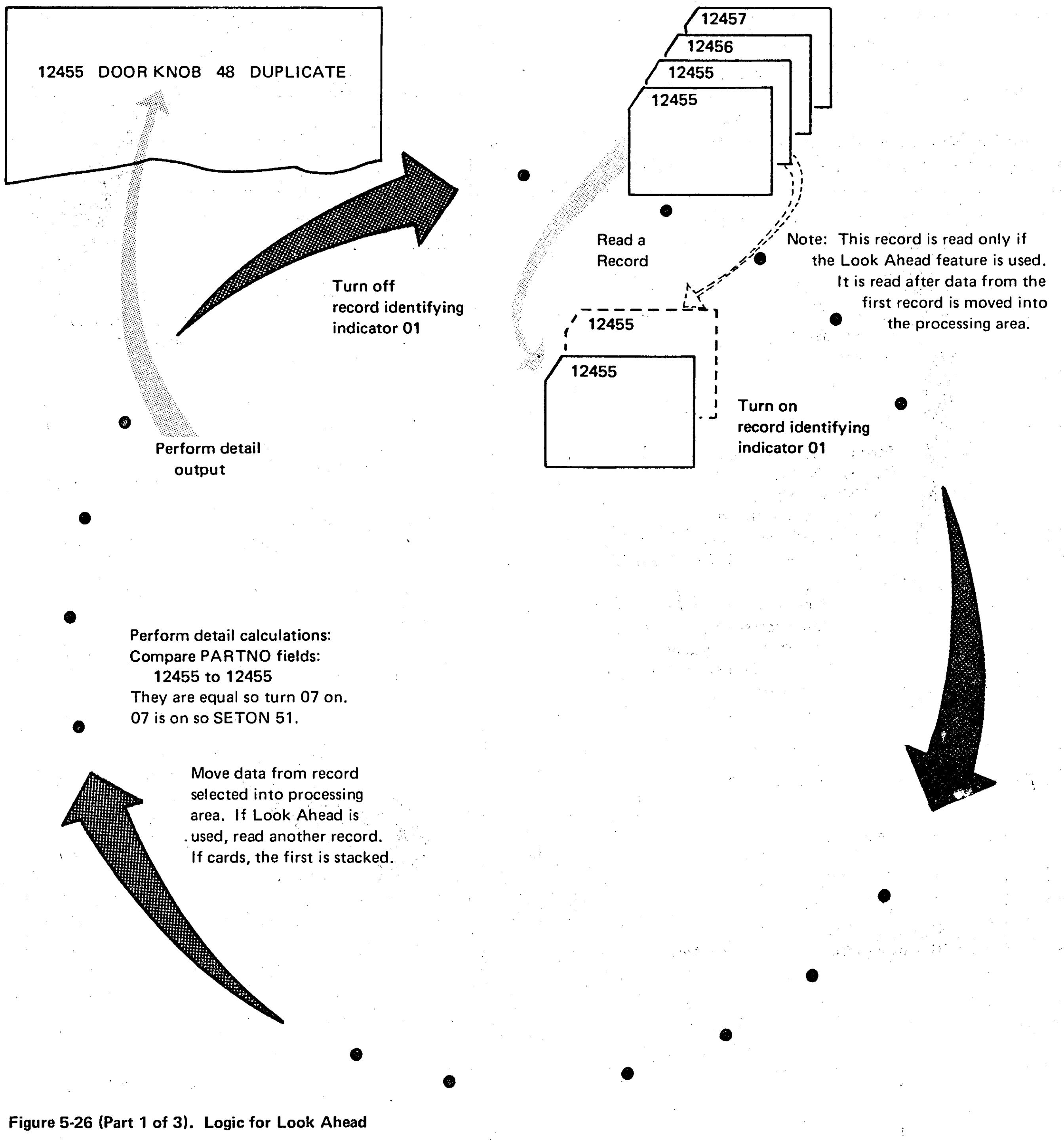
RPG OUTPUT SPECIFICATIONS

Line	Form Type	Filename	Type	Space Before	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	D	2	Ø1			
02	O					PARTNO	25	
03	O					DESC	6Ø	
04	O					ONHAND1	75	
Ⓑ	O				Ø7		9Ø	'DUPLICATE'

Figure 5-25. Using Information from the Look Ahead Field to Check for Duplicates

When PARTNO equals NEXTNO, 07 turns on. This, in turn, causes indicator 51, which is used to indicate that a duplicate record is processed, to turn on. During the next program cycle, the compare does not indicate duplicates; therefore 07 is not on. But 51 *is* on, meaning that the record being processed is a duplicate since the part number on it matched the part number on the previous record. Therefore, 07 is set on. Remember 07 conditions those output operations which are to be done for duplicates.

Indicator 52 is set on in line 03 to indicate that the last duplicate record is being processed. Indicator 52 then conditions line 04 so that indicator 51 will be set off and not indicate duplicates in the following cycle. Figure 5-26 shows the program logic for this job.



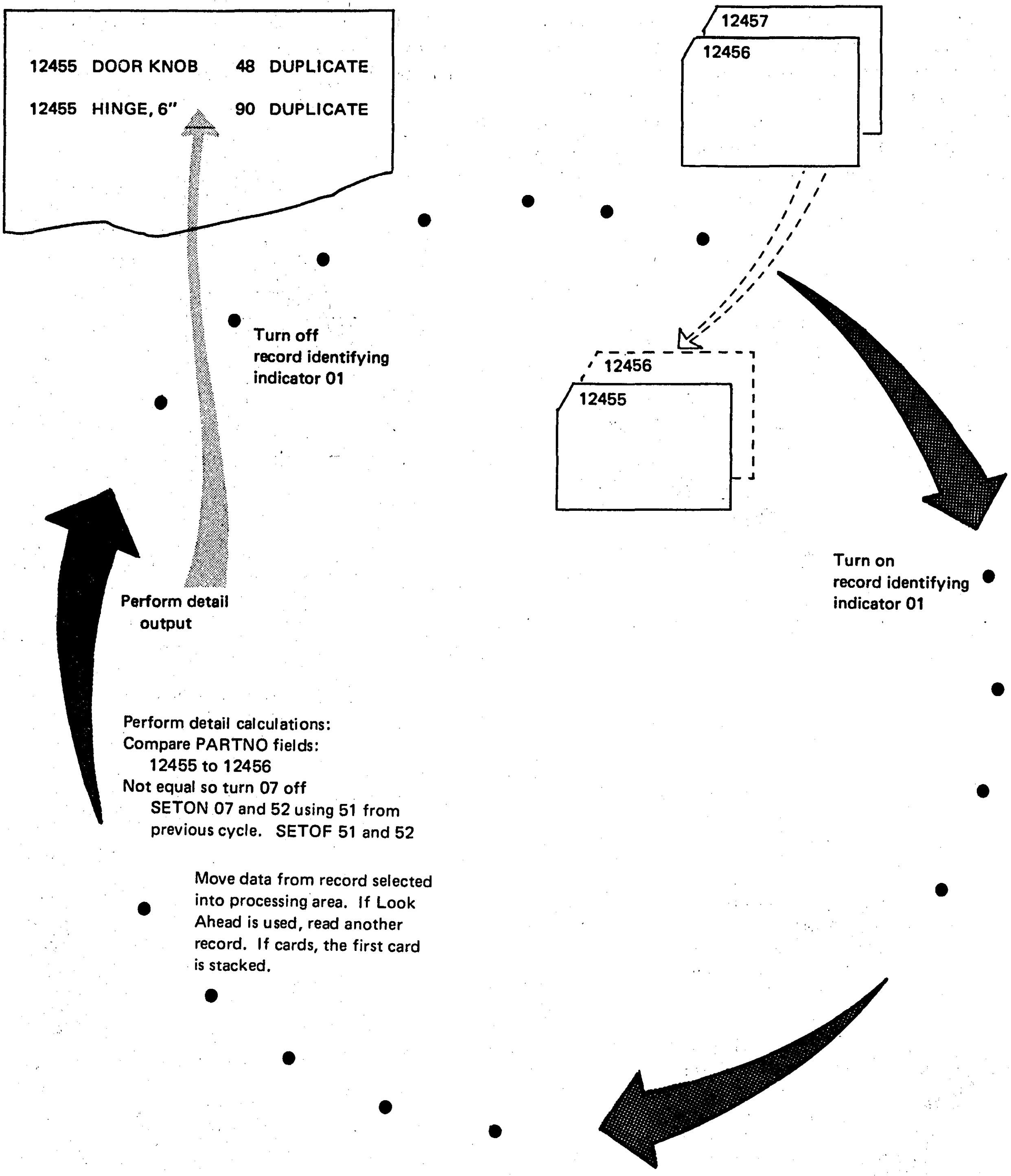
12455 DOOR KNOB 48 DUPLICATE
12457
12456
12455
12455
Read a Record
12455
12455
Turn on record identifying indicator 01
Turn off record identifying indicator 01
Perform detail output

Note: This record is read only if the Look Ahead feature is used. It is read after data from the first record is moved into the processing area.

Move data from record selected into processing area. If Look Ahead is used, read another record. If cards, the first is stacked.

Perform detail calculations:
Compare PARTNO fields:
12455 to 12455
They are equal so turn 07 on.
07 is on so SETON 51.

Figure 5-26 (Part 1 of 3). Logic for Look Ahead



```
12455  DOOR KNOB    48  DUPLICATE

12455  HINGE, 6"    90  DUPLICATE
```

Turn off record identifying indicator 01

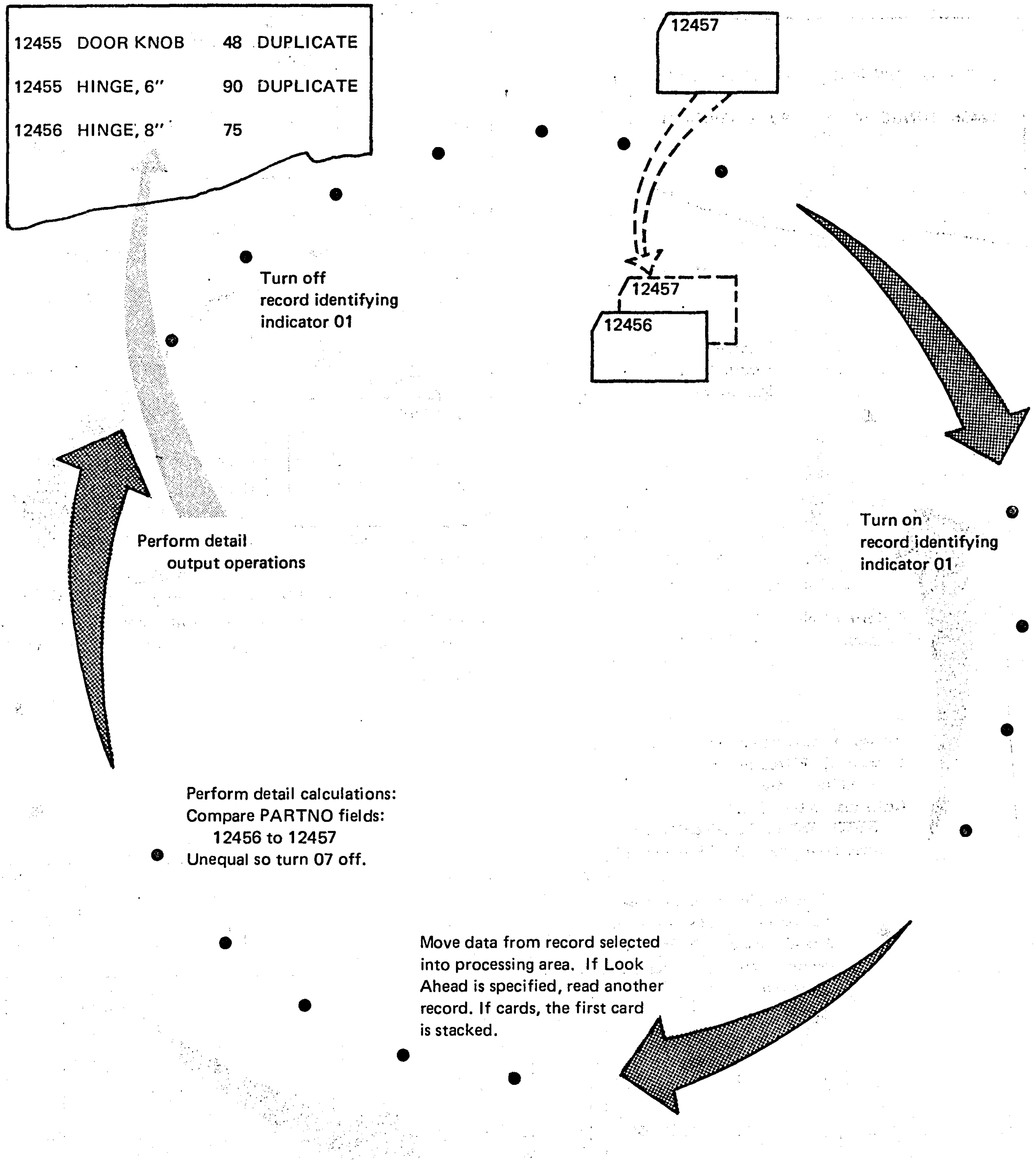
Turn on record identifying indicator 01

Perform detail output

Perform detail calculations:
Compare PARTNO fields:
12455 to 12456
Not equal so turn 07 off
SETON 07 and 52 using 51 from previous cycle. SETOF 51 and 52

Move data from record selected into processing area. If Look Ahead is used, read another record. If cards, the first card is stacked.

Figure 5-26 (Part 2 of 3). Logic for Look Ahead



```
12455  DOOR KNOB    48  DUPLICATE
12455  HINGE, 6"    90  DUPLICATE
12456  HINGE, 8"    75
```

Turn on record identifying indicator 01

Move data from record selected into processing area. If Look Ahead is specified, read another record. If cards, the first card is stacked.

Perform detail calculations:
Compare PARTNO fields:
12456 to 12457
Unequal so turn 07 off.

Perform detail output operations

Turn off record identifying indicator 01

Figure 5-26 (Part 3 of 3). Logic for Look Ahead

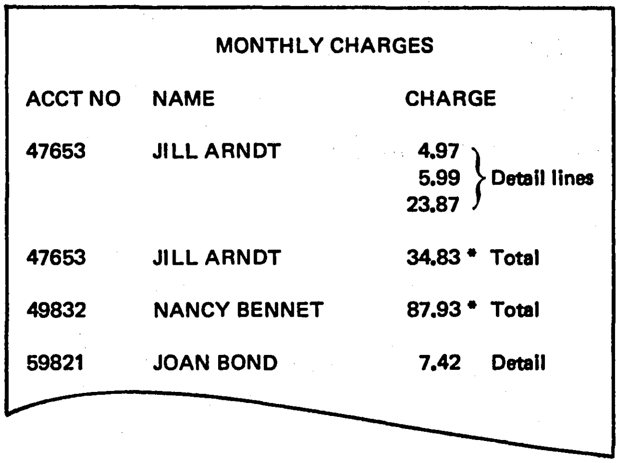
Doing Special Operations When There is Only One Record in a Group

It is often important to know if and when you are processing the only record in a group. The program described in the following paragraphs is such a case.

A report is prepared showing charges made by customers during the month (Figure 5-27). The input file is organized in ascending order by customer number. During the month some customers will have made one charge; others several.

When only one charge is made per customer, the total line is nearly a duplicate of the only detail line. In this case, you do not need to print both the detail and total line because the total line will do.

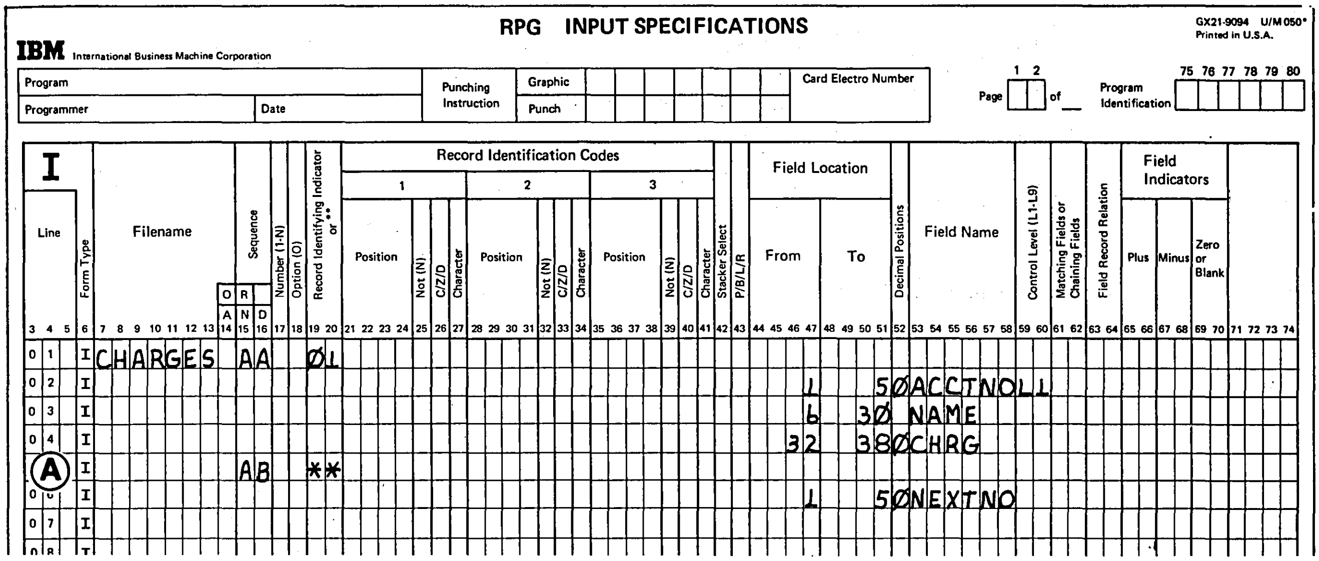
But how will you know during any one program cycle whether the current record is the only one in a group? You can find out by looking at information on the next record. Remember, any time information from the next record is necessary to determine how to process the current record, you must use the look ahead feature. Account number is established as a look ahead field in this program. Any look ahead field specified applies to all record types. Thus each record read contains information that will be looked at before the record itself is processed. By looking ahead into this field you will know whether or not the next record to be processed is part of a new group.



```
                      MONTHLY CHARGES

ACCT NO    NAME                      CHARGE

47653      JILL ARNDT                 4.97 )
                                      5.99  } Detail lines
                                     23.87 )

47653      JILL ARNDT                34.83 *  Total

49832      NANCY BENNET              87.93 *  Total

59821      JOAN BOND                  7.42    Detail
```

Figure 5-27. Format of Monthly Charges Report

Whenever a record is read, the current ACCTNO field is compared to the one coming up. If the fields are equal, you know you are processing a record that is not the only one in a group. Therefore, a detail line should be printed. If the ACCTNO fields are not equal and this is the first time the present account number has been encountered, the current record is the only one in the group, and the detail line should not print. Figure 5-28 shows the specifications for the program.



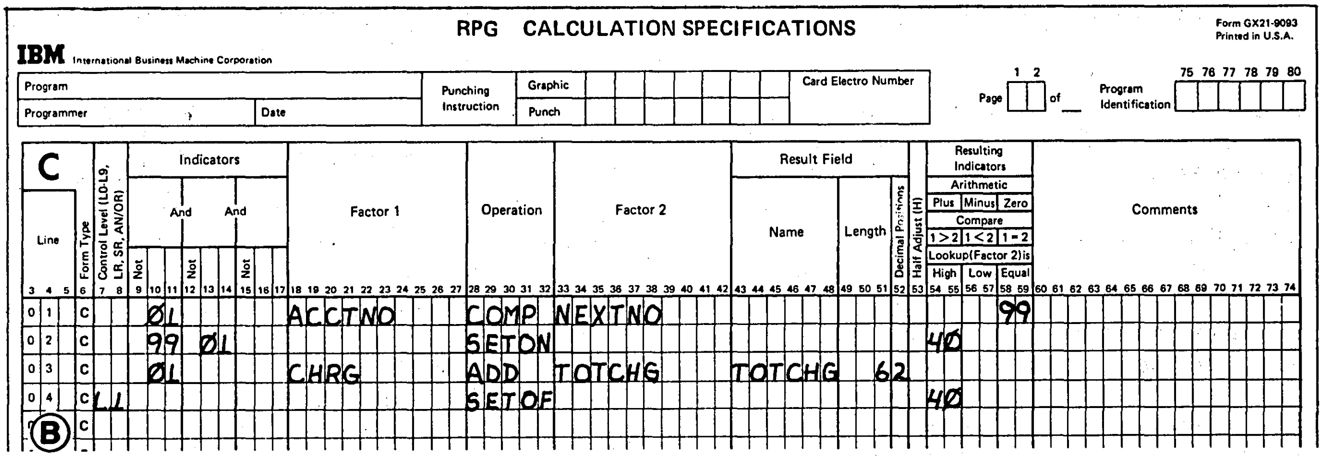
IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M050*
Printed in U.S.A.

```
01 ICHARGES AA  Ø1
02 I                                        1   5ØACCTNOL1
03 I                                        6  3Ø NAME
04 I                                       32  38ØCHRG
A  I        AB  **
06 I                                        1   5ØNEXTNO
07 I
08 I
```

Figure 5-28 (Part 1 of 2). Using Look Ahead to Determine When There is Only One Record in a Group



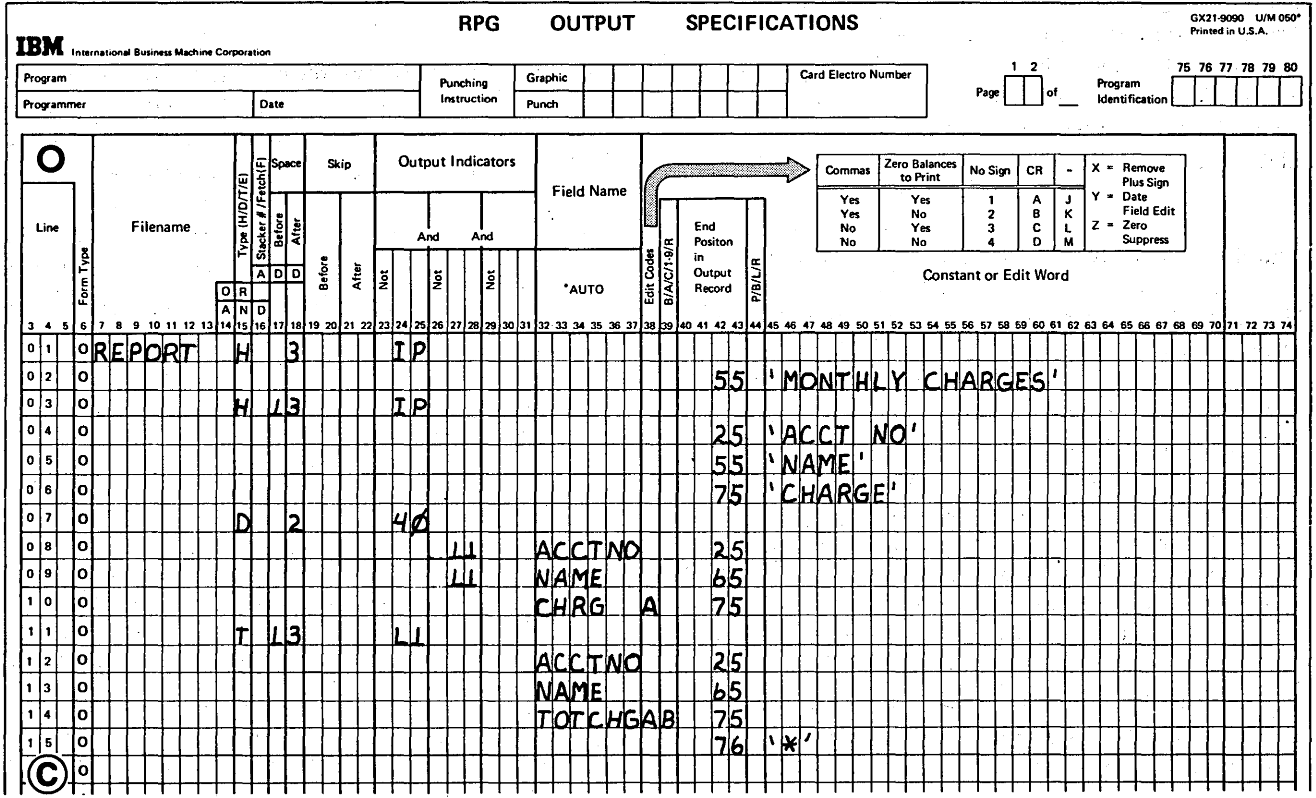
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	And	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Plus / High / 1>2	Minus / Low / 1<2	Zero / Equal / 1=2	Comments
01	C		ØL			ACCTNO	COMP	NEXTNO							99	
02	C		99	ØL			SETON						4Ø			
03	C		ØL			CHRG	ADD	TOTCHG	TOTCHG	6	2					
04	C	L1					SETOF						4Ø			
B	C															



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker #/Fetch (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	And	And	Field Name (*AUTO)	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	REPORT	H			3			1P								
02	O														55		'MONTHLY CHARGES'
03	O		H		1	3			1P								
04	O														25		'ACCT NO'
05	O														55		'NAME'
06	O														75		'CHARGE'
07	O		D			2			4Ø								
08	O									L1		ACCTNO			25		
09	O									L1		NAME			65		
10	O											CHRG	A		75		
11	O		T		1	3			L1								
12	O											ACCTNO			25		
13	O											NAME			65		
14	O											TOTCHG	A	B	75		
15	O														76		'*'
C	O																

Figure 5-28 (Part 2 of 2). Using Look Ahead to Determine When There is Only One Record in a Group

Doing Special Operations for the Last Record in a Group

In some programs, it may be necessary to do special operations on the last record of a control group. This is because, unless the last record in the control group is of a different type (have different record identification), it is impossible to know when you are processing the last record in the group. When all records are of the same type, you have to know what is on the next record before you know whether or not you are processing the last record in the group. To look at information in the next record, you must use the look ahead feature.

Figure 5-29 shows four records which are to be processed. The first three belong to one control group; the fourth is the beginning of the next group. The last record of the group (the third record in this case) requires special processing. In order to know when the last record in the group is to be processed, you must look at the account number in the next record. When it is different, you know that the last record in the group is being processed.

Additional Points to Consider About Look Ahead

You must consider the following things when you are planning to use look ahead:

- When look ahead is used with a combined or update file, and that file is the only input file in the program, the field looked at is not on the next record, but on the record currently being processed. Therefore, there is little use for look ahead with update or combined files in a single file program. In a program with multiple input-type files, look ahead fields can be useful in update and combined files. For example, if two files with look ahead fields are being processed — an input file and a combined file (or update file) — look ahead fields in the combined file are available as the input file is being processed (see Figure 5-22).

- Look ahead is never used with chained, demand, or output files.

- Only one look ahead record type specification may be used for a file. There may be several fields listed under that one record type specification however.

- Any look ahead fields specified apply to all types of records in the file. Therefore, all records read from the file will be treated as if they have look ahead fields.

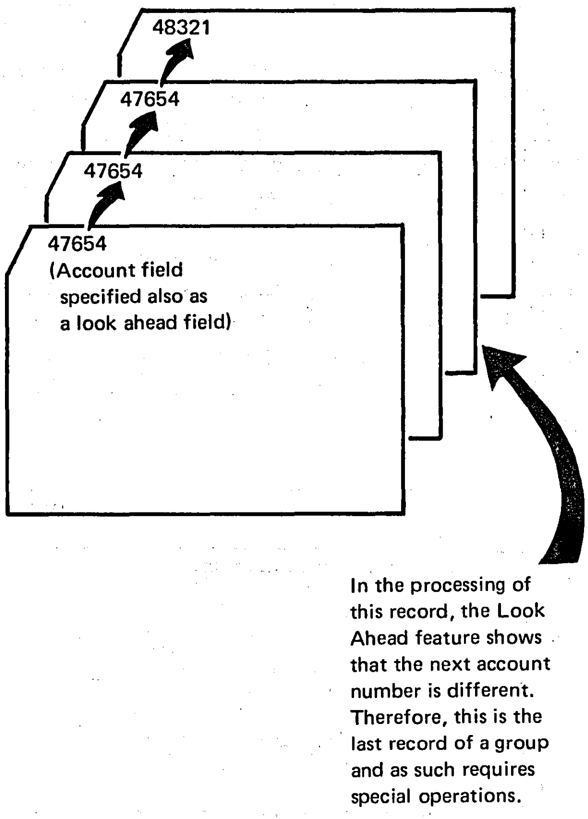

48321
47654
47654
47654
(Account field specified also as a look ahead field)
In the processing of this record, the Look Ahead feature shows that the next account number is different. Therefore, this is the last record of a group and as such requires special operations.

Figure 5-29. Using Look Ahead to Find Last Record in a Group

MOVING DATA

You can instruct the program to manipulate data in many different ways. You can cause data to be added, subtracted, multiplied, compared, tested or divided. You can also cause data to be moved. When data is moved, a copy of the data in one field is transferred to another field. In the process of transfer, it may or may not be changed depending upon your specifications.

You may wish to move data for many reasons, including:

1. To save information from a field.
2. To separate one field into 2 or more parts.
3. To change a numeric field into an alphameric field or vice versa.

The preceding topics will be discussed later in this section. First, however, you must learn how to use the move operation codes.

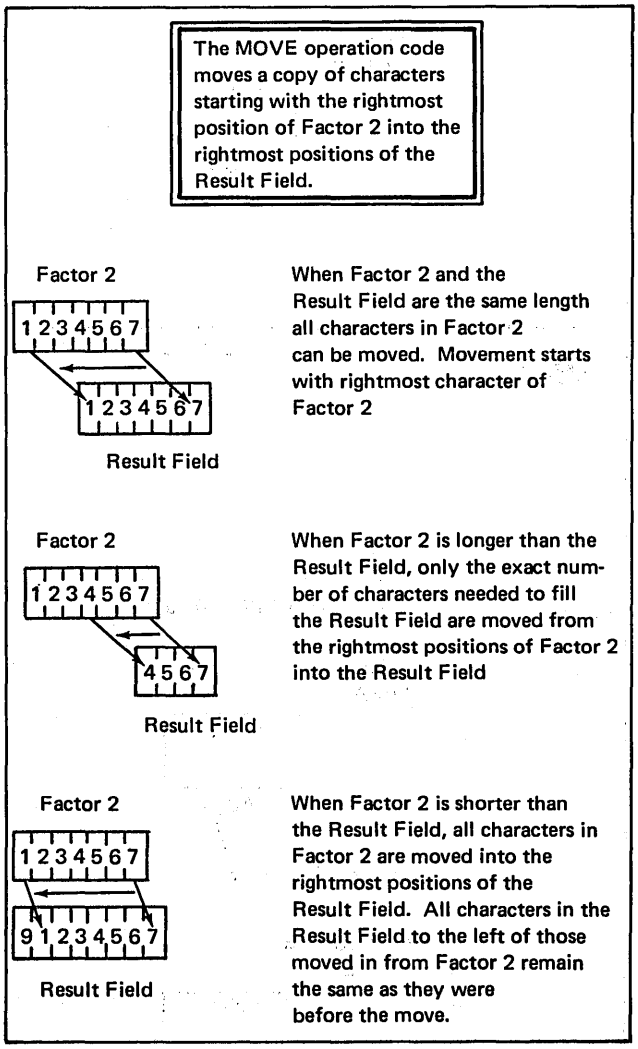
Specifications for Moving Data

Two operation codes can be used to move data: MOVE or MOVEL. For both operations Factor 2 and the Result Field are always used. Factor 2 may be either a field or constant. Any conditioning indicators may be used. However, Factor 1 and resulting indicators may not be specified.

The MOVE operation code moves a copy of characters starting from the rightmost position of Factor 2 into the rightmost positions of the Result Field. As a result of the move, the contents of the Result Field are changed, but the contents of Factor 2 remain the same. Figure 5-30 illustrates the MOVE operation.

If the Result Field is the same length as Factor 2, all characters in Factor 2 are transferred. However, if the Result Field is shorter than Factor 2, only the number of characters needed to fill the Result Field are transferred. On the other hand, if the Result Field is longer than Factor 2, all characters in Factor 2 are moved to the rightmost positions of the Result Field. The excess leftmost characters of the Result Field remain unchanged.

The MOVEL operation is just the reverse of the MOVE operation; it moves a copy of the characters starting from the *leftmost* position of Factor 2 into the leftmost positions of the Result Field. Figure 5-31 illustrates the MOVEL operation.



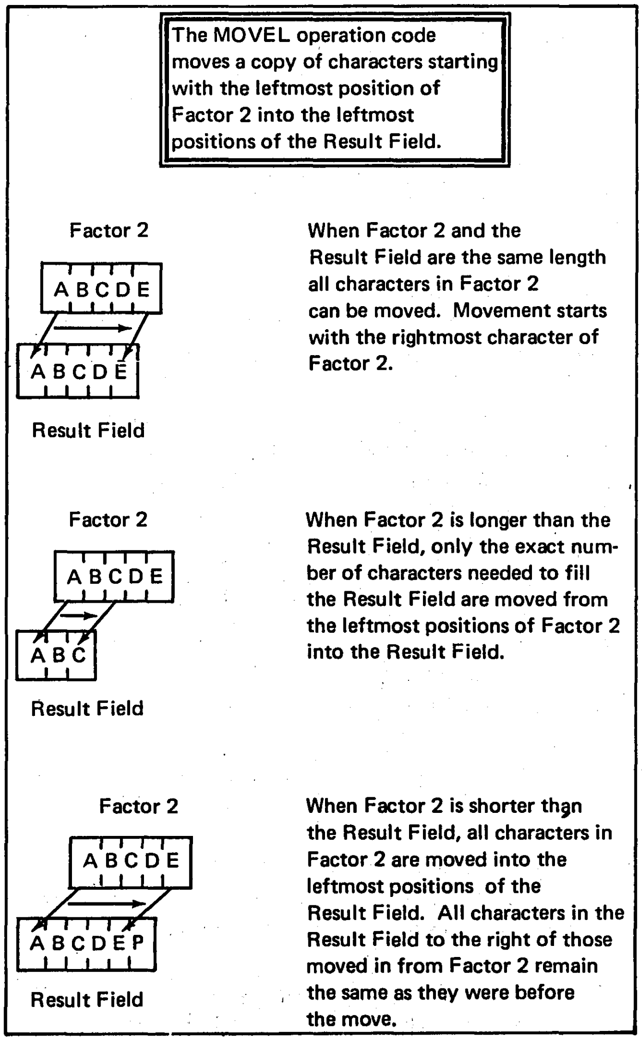
The MOVE operation code moves a copy of characters starting with the rightmost position of Factor 2 into the rightmost positions of the Result Field.

When Factor 2 and the Result Field are the same length all characters in Factor 2 can be moved. Movement starts with rightmost character of Factor 2

When Factor 2 is longer than the Result Field, only the exact number of characters needed to fill the Result Field are moved from the rightmost positions of Factor 2 into the Result Field

When Factor 2 is shorter than the Result Field, all characters in Factor 2 are moved into the rightmost positions of the Result Field. All characters in the Result Field to the left of those moved in from Factor 2 remain the same as they were before the move.

Figure 5-30. MOVE Operation Code



The MOVEL operation code moves a copy of characters starting with the leftmost position of Factor 2 into the leftmost positions of the Result Field.

When Factor 2 and the Result Field are the same length all characters in Factor 2 can be moved. Movement starts with the rightmost character of Factor 2.

When Factor 2 is longer than the Result Field, only the exact number of characters needed to fill the Result Field are moved from the leftmost positions of Factor 2 into the Result Field.

When Factor 2 is shorter than the Result Field, all characters in Factor 2 are moved into the leftmost positions of the Result Field. All characters in the Result Field to the right of those moved in from Factor 2 remain the same as they were before the move.

Figure 5-31. MOVEL Operation Code

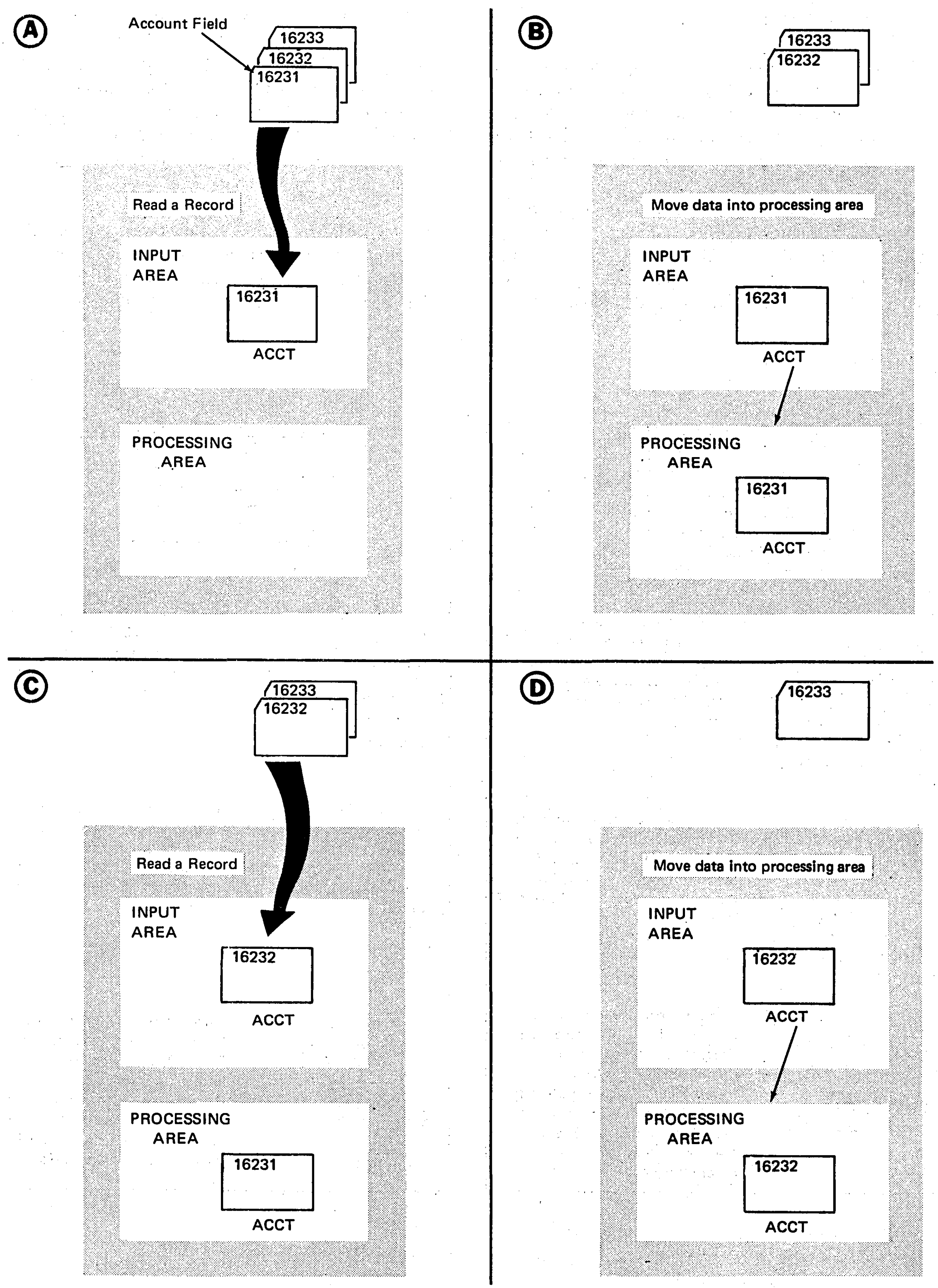


Figure 5-32. Data in Storage

Saving Information From a Field by Move Operations

Any information in fields specified as input fields is normally only available for one program cycle. Each time a record is read, information from the fields on the new record replaces that which was there from the previous record (see Figure 5-32).

There are times, however, when you wish to save information from a record so that you can have it available in the next program cycle. For instance, you may want to check the contents of a field in order to determine if a file is in proper sequence or if it has duplicate entries. In order to do this, you must have the data from two fields, the field from the record just read and the field from the record read in the previous cycle.

Consider the problem of checking the sequence of, and finding duplicate entries for, the file shown in Figure 5-33. Sequence is to be based on ACCT (account number) and must be ascending. Any duplicate or out-of-sequence records are to be flagged.

The program must compare the ACCT fields from two records: the record currently being processed and the previous record. The current account field should always be higher than the previous account field. Would the specifications in Figure 5-34 do the job? They *would not,* because in one program cycle the field named ACCT always has the same information in it. This information is taken from the record just read. Just because you use the name twice, you won't get two different ACCT fields.

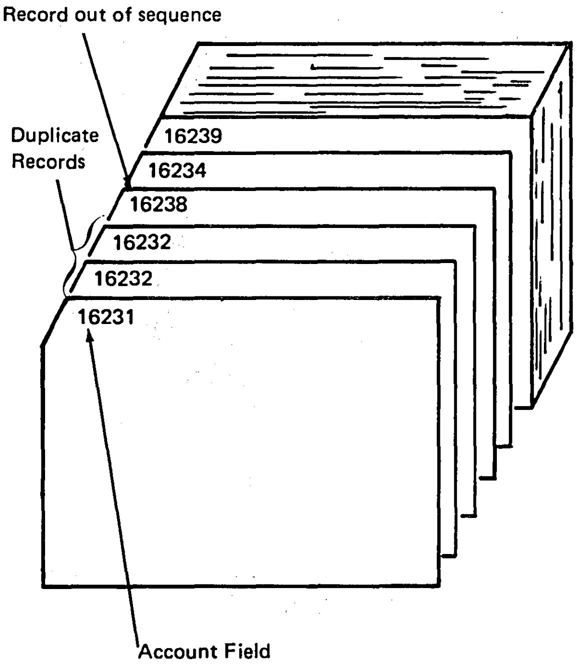
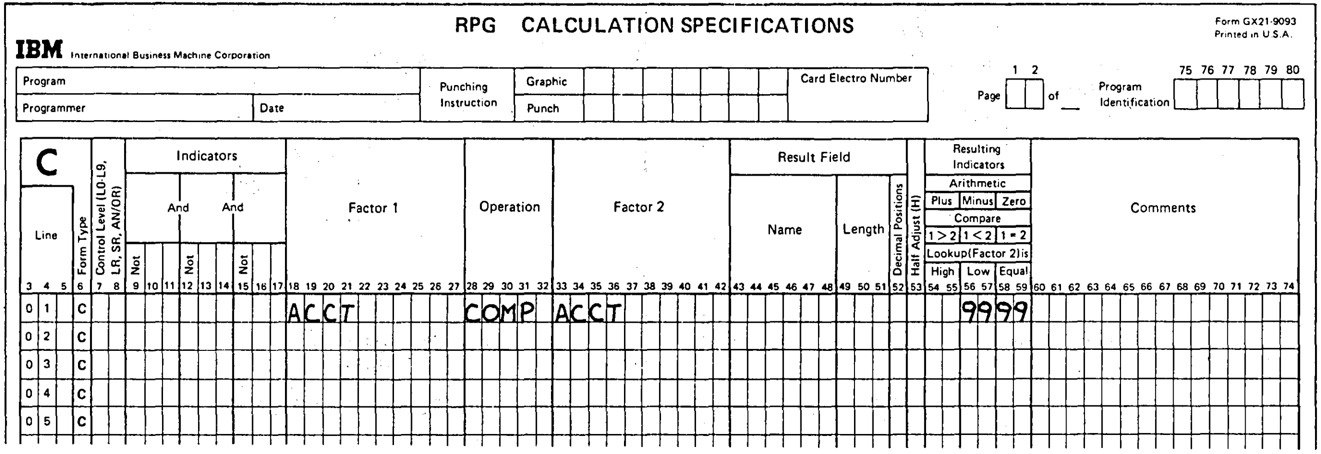


Figure 5-33. Input Field Sequence



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page □□ of __
Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	High (1>2)	Low (1<2)	Equal (1=2)	Comments
01	C			ACCT	COMP	ACCT						99	99	
02	C													
03	C													
04	C													
05	C													

Figure 5-34. Sequence Checking (Incomplete Specifications)

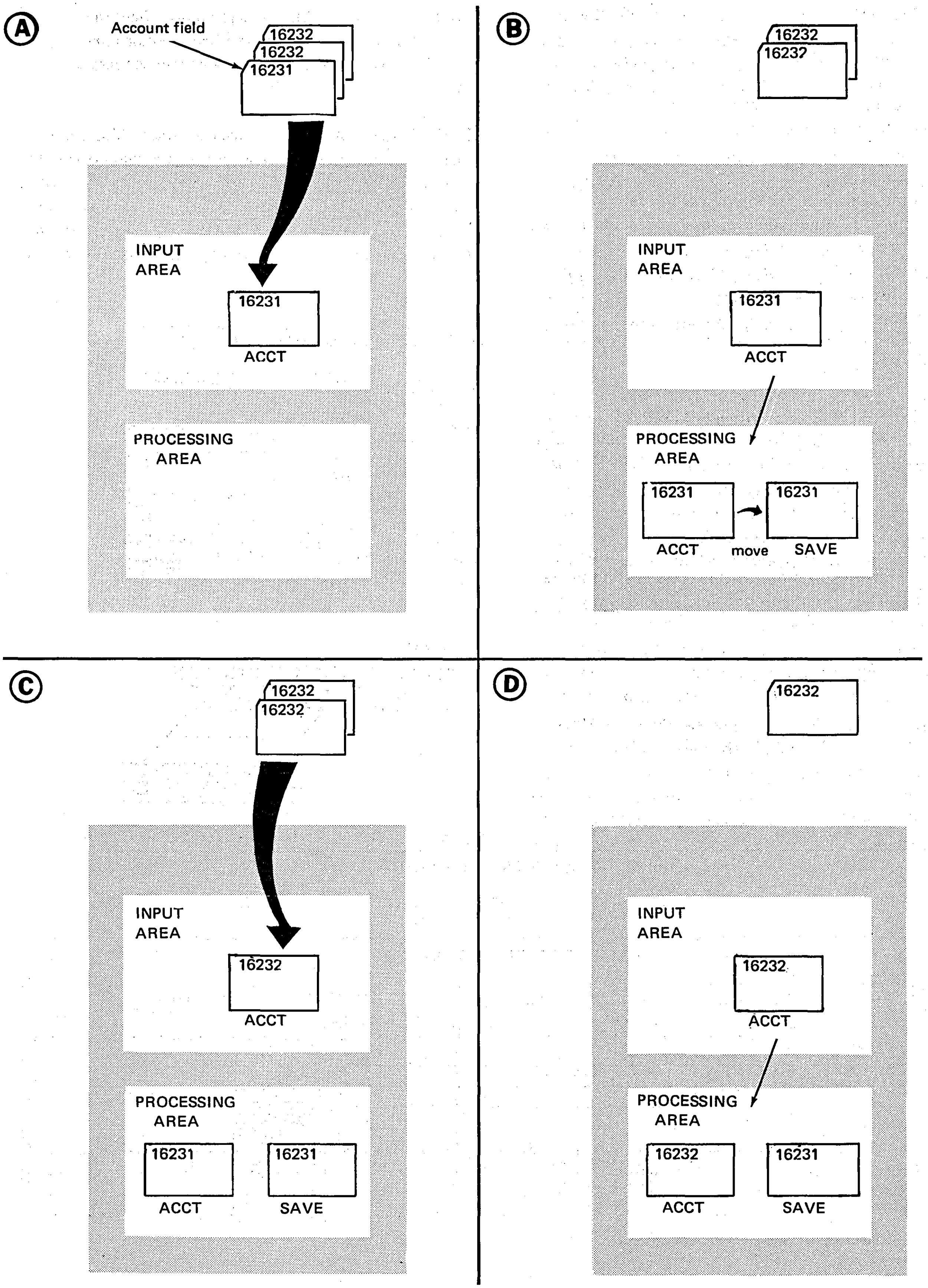


Figure 5-35. Moving Data to Save It

Therefore, each time a record is read, you must save the information from the ACCT field so that it is not destroyed when another record is read. This you can do by moving ACCT into another field. (Figure 5-35 illustrates this concept.)

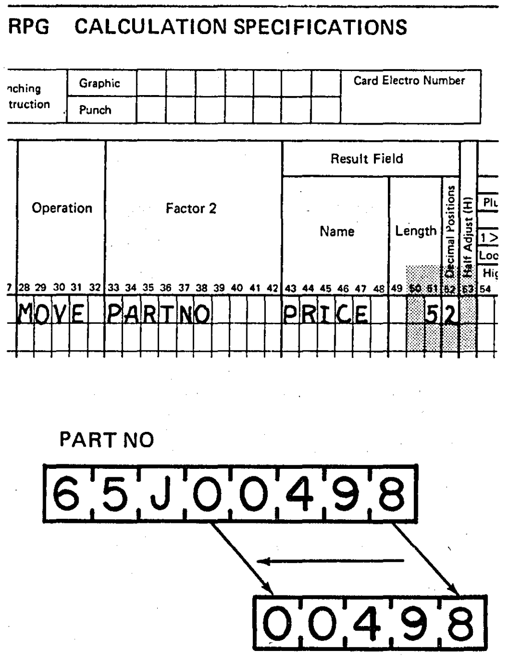
Figure 5-36 shows the specifications needed to do the job. Assume that you move ACCT into a field called SAVE. The first step is to compare ACCT with SAVE (which contains the previous ACCT data). The second step is to move ACCT into SAVE. In the first program cycle, SAVE will contain all zeros since all numeric fields are set up with zeros before the first record is read. In the next cycle, and all cycles thereafter, SAVE will contain the ACCT field from the previous record.

Maybe you are wondering why you would ever want to sequence check by calculations instead of using the RPG II automatic sequence checking function which is done merely by specifying a match field. The answer is that, with RPG II automatic sequence checking, any out-of-sequence card will cause a *halt.* If you do not wish to halt, but merely wish to indicate out-of-sequence or duplicate cards, then you must do your own sequence checking. You could also use the look ahead feature, since both look ahead and the use of moves in calculations give essentially the same results.

Separating One Field Into Two Parts

A company has designed its part numbers to contain two different kinds of information. The basic part number is contained in the first three characters. The remaining five characters contain the price. For example, in the part number 65J00498, the basic part number is 65J and the price is $4.98. When preparing invoices, it is necessary to multiply unit price times quantity to find the total price. You don't want to multiply the whole part number field times quantity just because the part of the field contains unit price. Somehow, you must separate price from the rest of the field.

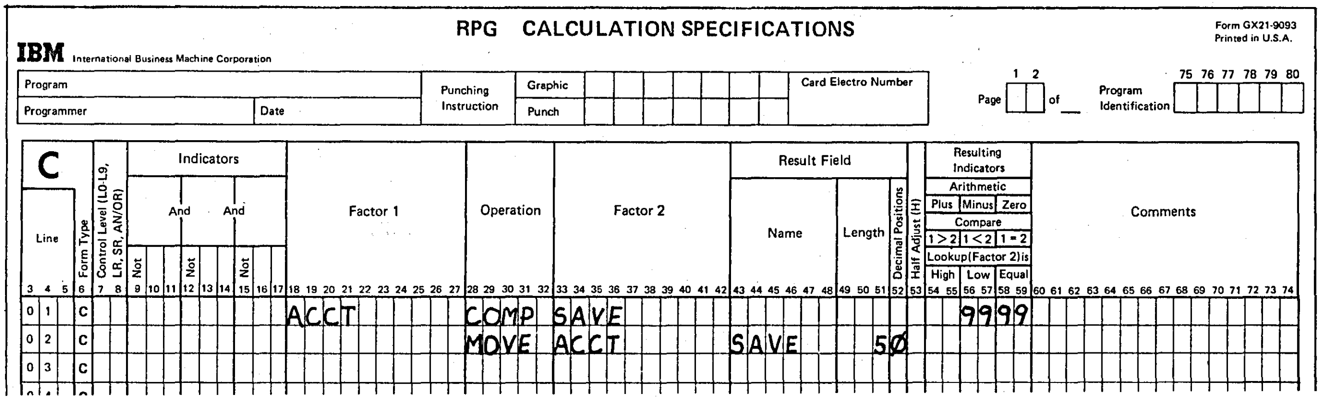
To do this you again use a move operation. You cannot move the whole field into another field as was done in the previous example. This merely creates a second field identical to the first. You want only the last five characters. Therefore, you must move the field into a 5-position field. This will limit the move to five characters (see Figure 5-37).



RPG CALCULATION SPECIFICATIONS

Operation	Factor 2	Result Field Name	Length	Decimal Positions
MOVE	PARTNO	PRICE	5	2

Figure 5-37. Separating the Price from the Part Number (Using MOVE)



RPG CALCULATION SPECIFICATIONS

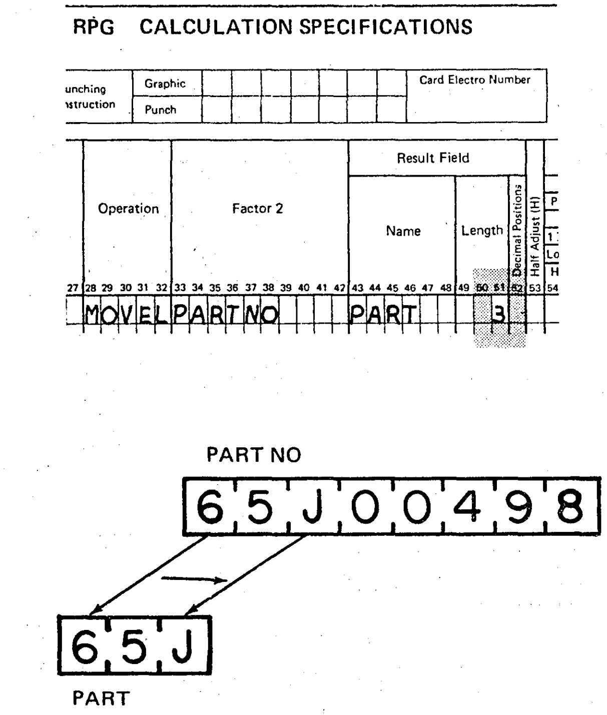
IBM International Business Machine Corporation

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	High	Low	Equal
01	C	ACCT	COMP	SAVE					99	99
02	C		MOVE	ACCT	SAVE	5	Ø			
03	C									

Figure 5-36. Sequence Checking (Correct Specifications)

In this case, the MOVE operation was used. Why wouldn't the MOVEL operation work as well? Couldn't you move left into a 3-character field to separate the price from the rest of the part number? Remember that a move leaves the original field as it was before the move. MOVEL does not *remove* the first three characters from the original PARTNO field. It merely copies them. You still would not have the unit cost by itself. But you would have the part number by itself (see Figure 5-38).



RPG CALCULATION SPECIFICATIONS

Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)
MOVEL	PARTNO	PART	3		

Figure 5-38. Separating the Part from the Part Number (Using MOVEL)

Changing Field Type (Alphameric or Numeric)

The part number field contains both numeric and alphabetic characters (see Figure 5-38). If you want the program to work with the zone portion of a character (as it must to get the letter J), you have to define the field as alphameric (blank in column 52 of the Input sheet). However, you cannot use an alphameric field in arithmetic calculations. To find total cost, you must multiply unit price by quantity. This is an arithmetic operation; thus the field must be numeric.

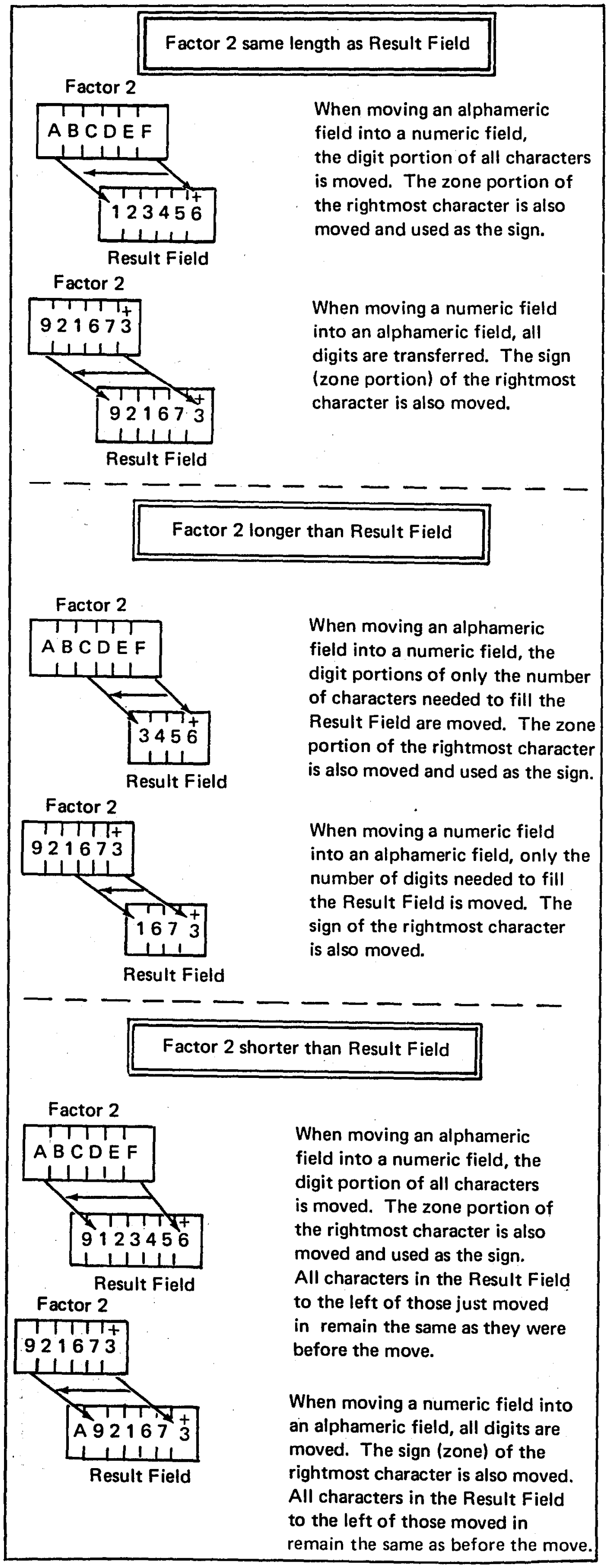
What do you do if you need one field to be defined as both alphameric and numeric? You could define the field twice — once as numeric and once as alphameric. Of course, you would have to use two names for the same field since every field defined for one type of record must have a unique name.

You may also use a move operation to change a numeric field into an alphameric field or vice versa. You can change fields by:

1. Moving an alphameric field named in Factor 2 into a numeric Result Field.

2. Moving a numeric field named in Factor 2 into an alphameric Result Field.

Figures 5-39 and 5-40 give the rules for and examples of the various types of moves you can make to change a field type. Figure 5-39 illustrates the MOVE operation and Figure 5-40 the MOVEL operation. If you do not understand results obtained in the low order positions, see the chapter entitled *Working With Data Structures.*



Factor 2 same length as Result Field

When moving an alphameric field into a numeric field, the digit portion of all characters is moved. The zone portion of the rightmost character is also moved and used as the sign.

When moving a numeric field into an alphameric field, all digits are transferred. The sign (zone portion) of the rightmost character is also moved.

Factor 2 longer than Result Field

When moving an alphameric field into a numeric field, the digit portions of only the number of characters needed to fill the Result Field are moved. The zone portion of the rightmost character is also moved and used as the sign.

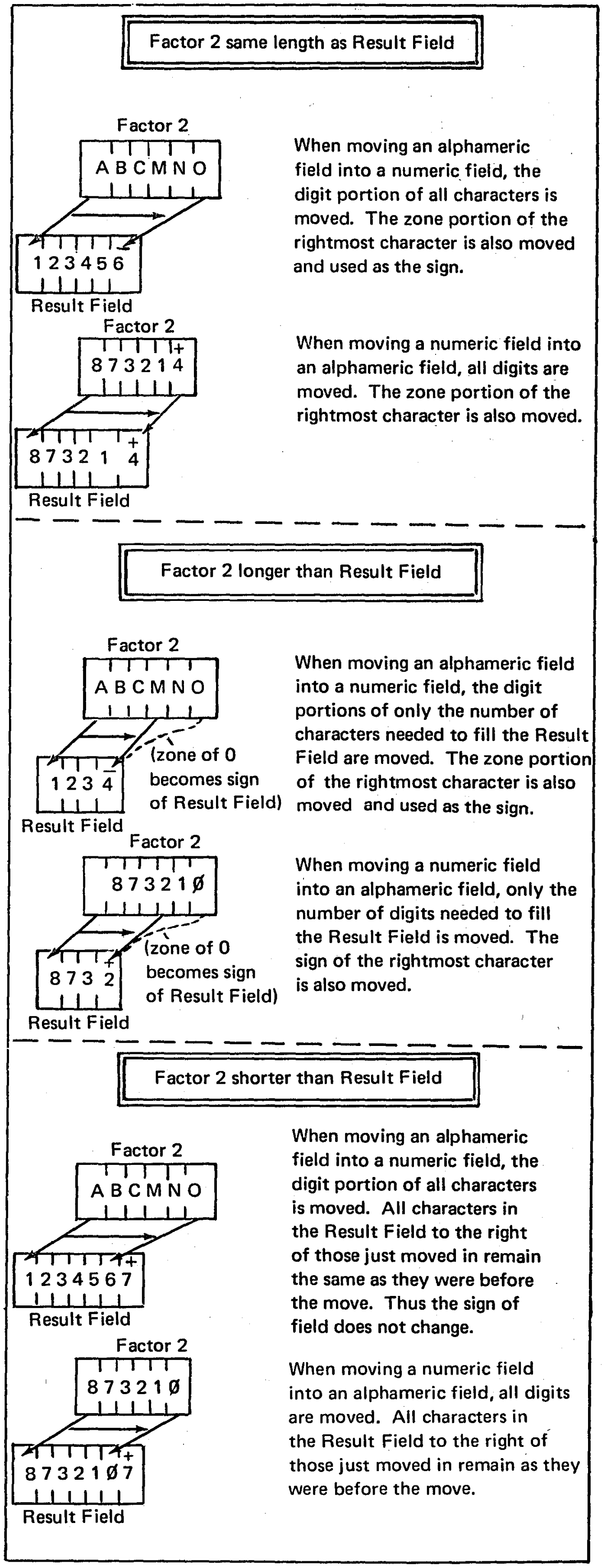
When moving a numeric field into an alphameric field, only the number of digits needed to fill the Result Field is moved. The sign of the rightmost character is also moved.

Factor 2 shorter than Result Field

When moving an alphameric field into a numeric field, the digit portion of all characters is moved. The zone portion of the rightmost character is also moved and used as the sign. All characters in the Result Field to the left of those just moved in remain the same as they were before the move.

When moving a numeric field into an alphameric field, all digits are moved. The sign (zone) of the rightmost character is also moved. All characters in the Result Field to the left of those moved in remain the same as before the move.

Figure 5.39. MOVE Operations Involving Fields of Various Lengths and Types



Factor 2 same length as Result Field

When moving an alphameric field into a numeric field, the digit portion of all characters is moved. The zone portion of the rightmost character is also moved and used as the sign.

When moving a numeric field into an alphameric field, all digits are moved. The zone portion of the rightmost character is also moved.

Factor 2 longer than Result Field

When moving an alphameric field into a numeric field, the digit portions of only the number of characters needed to fill the Result Field are moved. The zone portion of the rightmost character is also moved and used as the sign.

When moving a numeric field into an alphameric field, only the number of digits needed to fill the Result Field is moved. The sign of the rightmost character is also moved.

Factor 2 shorter than Result Field

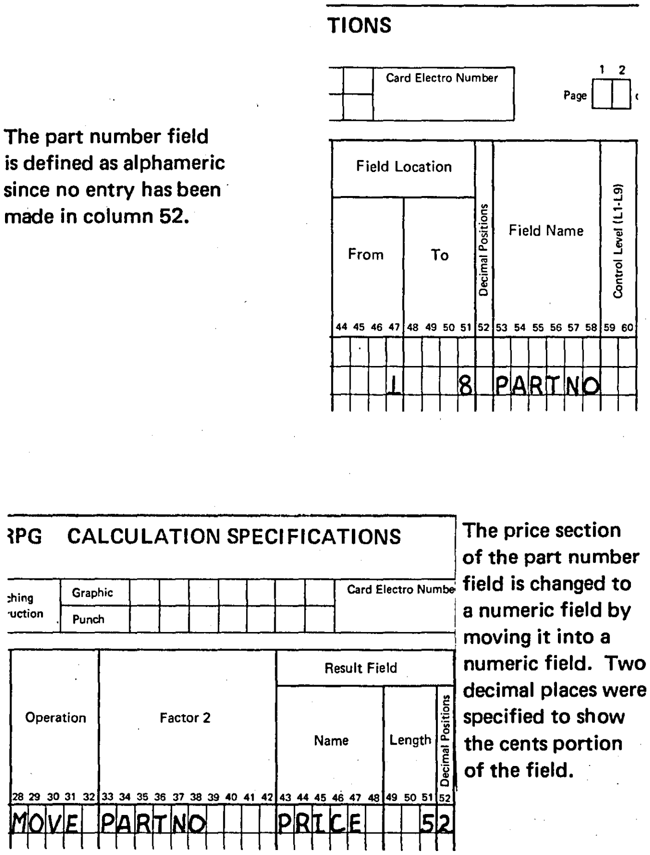
When moving an alphameric field into a numeric field, the digit portion of all characters is moved. All characters in the Result Field to the right of those just moved in remain the same as they were before the move. Thus the sign of field does not change.

When moving a numeric field into an alphameric field, all digits are moved. All characters in the Result Field to the right of those just moved in remain as they were before the move.

Figure 5-40. MOVEL Operations Involving Fields of Various Lengths and Types

In order for the letter in the part number ever to be read, compared, or printed, the field ..ust be defined as alphameric. When it is time to multiply price times quantity, the price portion of the field must be numeric. Therefore, when using the MOVE operation to separate the unit cost from the rest of the part number, you should, at the same time, change the alphameric unit price into a numeric unit price by moving it into a numeric field. To define a numeric field, you must specify decimal position along with field length (see Figure 5-41).

If the 5-character unit cost had preceded the part number (for example, 0049865J), you would then use the MOVEL operation to get the unit cost alone. Remember, however, that the zone of the rightmost character is used for the sign of the field (see Figure 5-40). The zone of the character J is a minus sign. The price will appear as negative. This you would not want.


TIONS
Card Electro Number
Page
1 2

The part number field is defined as alphameric since no entry has been made in column 52.

Field Location From (44 45 46 47)	Field Location To (48 49 50 51)	Decimal Positions (52)	Field Name (53 54 55 56 57 58)	Control Level (L1-L9) (59 60)
1	8		PARTNO	

RPG CALCULATION SPECIFICATIONS
Graphic
Punch
Card Electro Numbe

Operation (28 29 30 31 32)	Factor 2 (33 34 35 36 37 38 39 40 41 42)	Result Field Name (43 44 45 46 47 48)	Length (49 50 51)	Decimal Positions (52)
MOVE	PARTNO	PRICE	5	2

The price section of the part number field is changed to a numeric field by moving it into a numeric field. Two decimal places were specified to show the cents portion of the field.

Figure 5-41. Changing a Field by the MOVE Operation

When using move operations to change an alphameric to a numeric field, keep in mind the kind of sign (plus or minus) each character will give you: a - (minus sign), or J through R gives a minus sign; the rest give a positive sign. If you are aware of this, you will not get unexpected results. Since no sign is involved in an alphameric field, you don't need to worry about the sign when changing a numeric to an alphameric field.

BRANCHING IN CALCULATIONS

The detail and total operations written on the Calculation sheet are normally executed in the same order as they are written. For each record selected for processing, the detail operations are performed sequentially from beginning to end. If the record selected for processing causes a control break, the total operations are performed, in the order specified, before detail operations.

There are many times, however, when it is necessary that operations not be performed sequentially. For example, in one cycle you may wish to skip some calculations or to do others several times. In this section you will learn to alter the sequential processing of calculations using the most efficient coding.

Bypassing Calculations

So far you have been bypassing operations by the use of indicators. For each calculation conditioned by indicators, a check is made to see if the condition set by the indicators is satisifed. (When several sequential operations are conditioned by the same indicator(s), the test is only made on the first operation.) If the condition is satisfied, the operation is performed. Calculations are bypassed or omitted when conditions are not satisfied. When bypassing calculations in this way, the program has to check the conditions set for the operations to determine whether or not to do them. This requires time and storage space inside the computer.

Another way to bypass calculations is to branch around them. With the latter method, the indicator setting for each operation is not checked. When the branch is taken around operations, the operations are just skipped (see Figure 5-42, insert A).

Two operation codes are used for branching: GOTO and TAG. GOTO is the code which causes a branch to another spot in the calculations. The TAG operation gives the name and location of the spot to which the GOTO operation branches. GOTO causes a branch; the TAG code does nothing but act as a nametag.

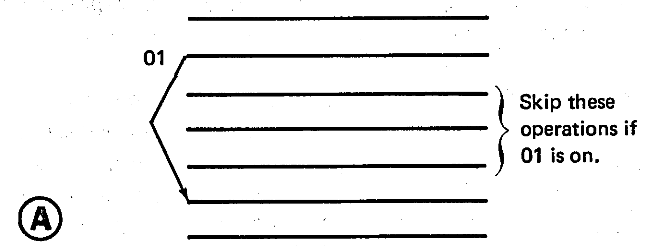
Figure 5-42, insert B, shows how GOTO and TAG are specified. GOTO signals a branch to the spot named in Factor 2. This name must also appear in the TAG statement, where it is entered in Factor 1. The rules for forming a name for GOTO and TAG are the same as those for forming any field name.

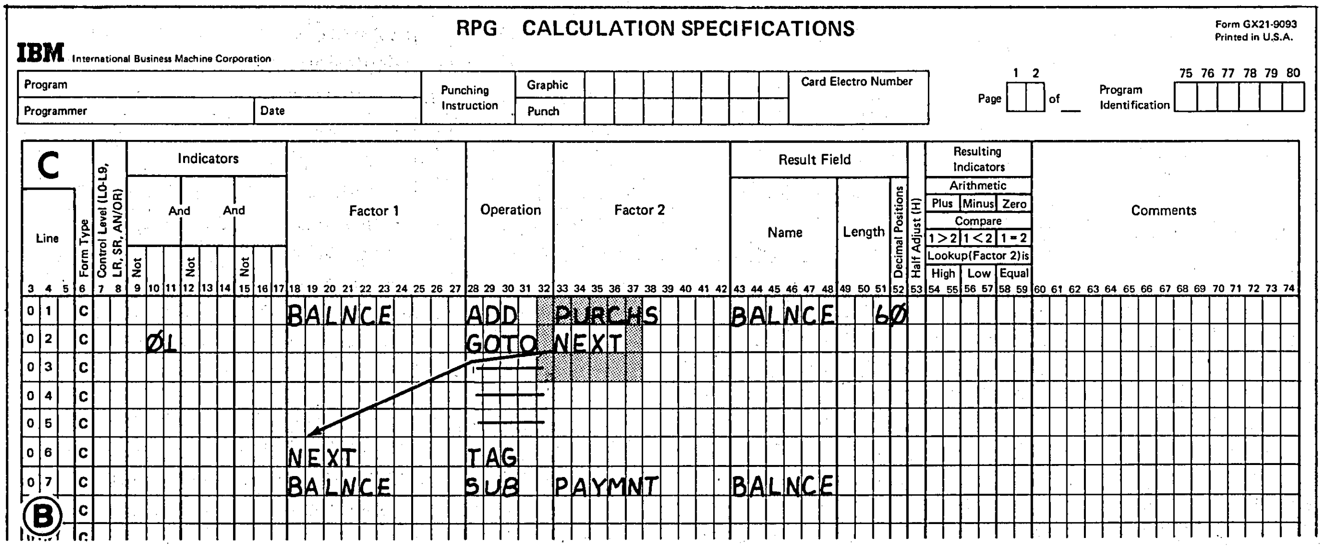
A GOTO statement can be conditioned by an indicator, but a TAG cannot. When a GOTO is not conditioned, a branch occurs in every program cycle.

In the example shown in Figure 5-42, the GOTO operation is done only when 01 is on. If the condition is satisfied, a branch is taken to that point in the program where the same NEXT is found. NEXT is the name of the TAG statement. Any operations between the GOTO statement and the TAG statement (those specified in lines 03-05) are skipped. TAG does nothing, so the next operation performed is the SUB instruction in line 07.

If branching were not done, the three operations skipped by the branch would have to be conditioned by NO1 so that they would not be done when 01 turned on. And, of course, a check would have to be made in each program cycle to determine if the operations should be done or not.

There are many situations in which branching will help you write more efficient and effective programs. The following sections will explain more fully the use of GOTO and TAG.




IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions
01	C		BALNCE	ADD	PURCHS	BALNCE	6	Ø
02	C	Ø1		GOTO	NEXT			
03	C							
04	C							
05	C							
06	C		NEXT	TAG				
07	C		BALNCE	SUB	PAYMNT	BALNCE		
	C							
	C							

Figure 5-42. Bypassing Calculations by Branching Around Them

Branching When Different Record Types Require Different Operations

When doing different operations for different record types, you use the record identifying indicators to show what operations should be done for each record type read (see Figure 5-43). When you have several record types and each type requires several operations, you can see that many conditioning indicators are necessary.

For situations like this, you can branch directly to the set of calculations which should be done for the record type just read. When those calculations are done, you can then branch to the end of all calculations. This eliminates checking operations to see if a set of calculations should be done for the record type being processed. In fact, record identifying indicators do not need to be specified for the individual operations. Figure 5-44 shows the recommended branching structure used for different record types which require different operations. Using this structure not only makes your programs more efficient but also makes them easier to understand and document.

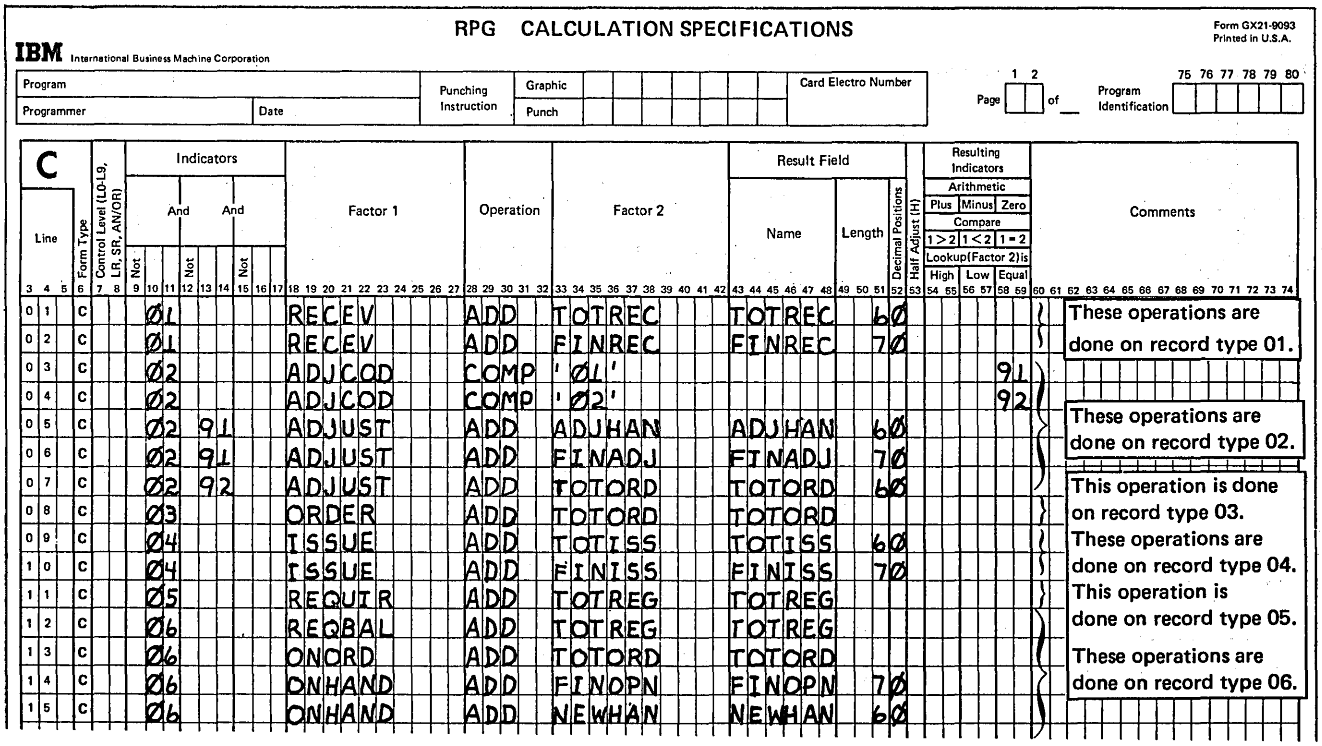
Consider the use of such a branching structure in a sales analysis program. Each day the manager of a retail store wishes to know total cash sales, total charge sales, and total refunds. The input file, arranged in ascending order by account number, contains four different record types:

1. Charge records record total charges and refunds (if any) for a particular account.

2. Payment records record any payments received. They are included in the file, but are not needed in this job.

3. Refund records record any refunds made for an account. No cash and charge sales were made by this customer.

4. Cash sales records record total cash sales and cash refunds (if any) for a particular account.



IBM International Business Machine Corporation

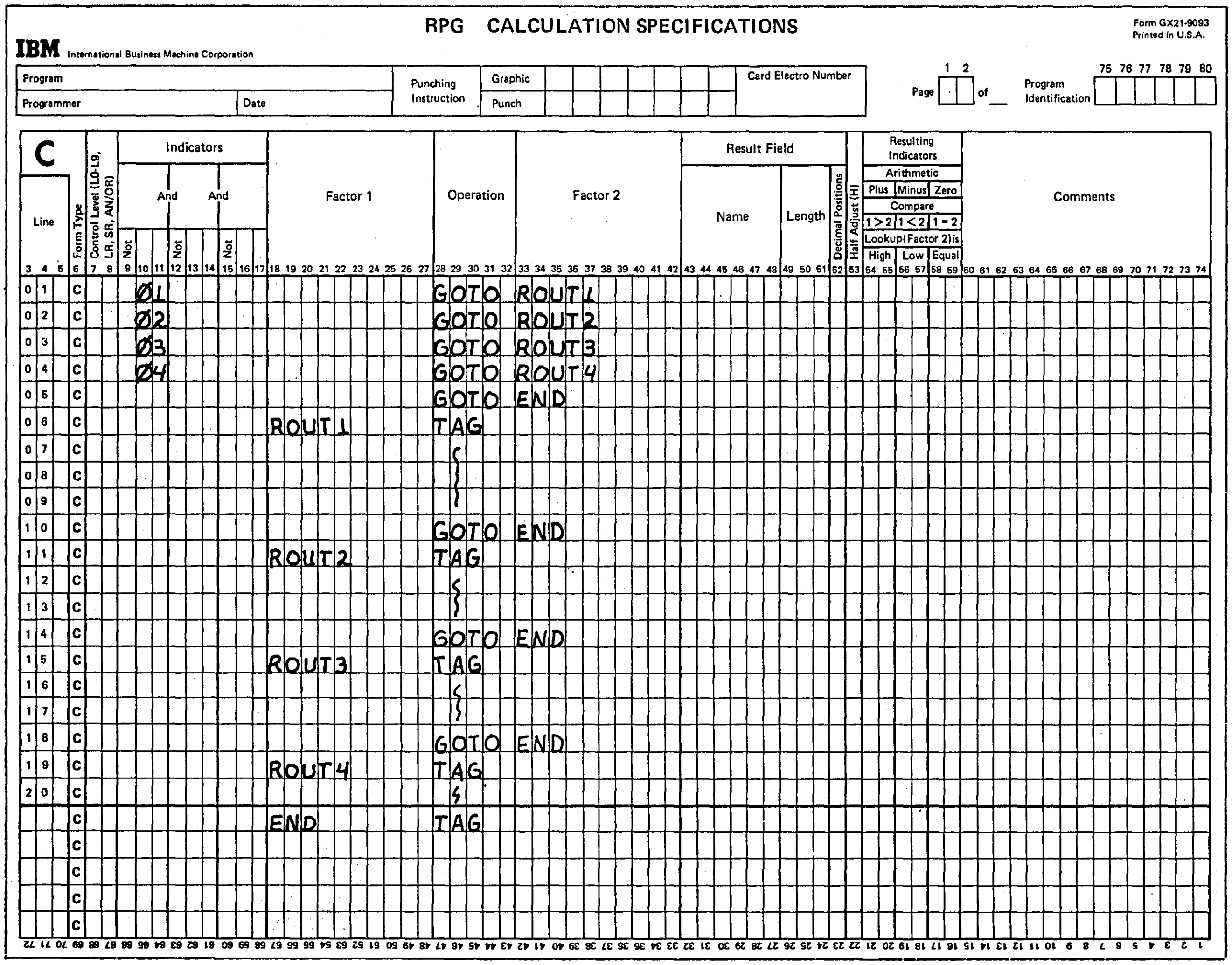
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page □□ of __
Program Identification

Line	Form Type	Control Level	Indicators (Not)	Indicators (And)	Indicators (And)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Plus / High	Minus / Low	Zero / Equal	Comments
01	C		Ø1			RECEV	ADD	TOTREC	TOTREC	6	Ø					These operations are
02	C		Ø1			RECEV	ADD	FINREC	FINREC	7	Ø					done on record type 01.
03	C		Ø2			ADJCOD	COMP	'Ø1'							91	
04	C		Ø2			ADJCOD	COMP	'Ø2'							92	
05	C		Ø2	91		ADJUST	ADD	ADJHAN	ADJHAN	6	Ø					These operations are
06	C		Ø2	91		ADJUST	ADD	FINADJ	FINADJ	7	Ø					done on record type 02.
07	C		Ø2	92		ADJUST	ADD	TOTORD	TOTORD	6	Ø					This operation is done
08	C		Ø3			ORDER	ADD	TOTORD	TOTORD							on record type 03.
09	C		Ø4			ISSUE	ADD	TOTISS	TOTISS	6	Ø					These operations are
10	C		Ø4			ISSUE	ADD	FINISS	FINISS	7	Ø					done on record type 04.
11	C		Ø5			REQUIR	ADD	TOTREG	TOTREG							This operation is
12	C		Ø6			REQBAL	ADD	TOTREG	TOTREG							done on record type 05.
13	C		Ø6			ONORD	ADD	TOTORD	TOTORD							These operations are
14	C		Ø6			ONHAND	ADD	FINOPN	FINOPN	7	Ø					done on record type 06.
15	C		Ø6			ONHAND	ADD	NEWHAN	NEWHAN	6	Ø					

Figure 5-43. Operations Performed for Different Record Types



IBM International Business Machine Corporation

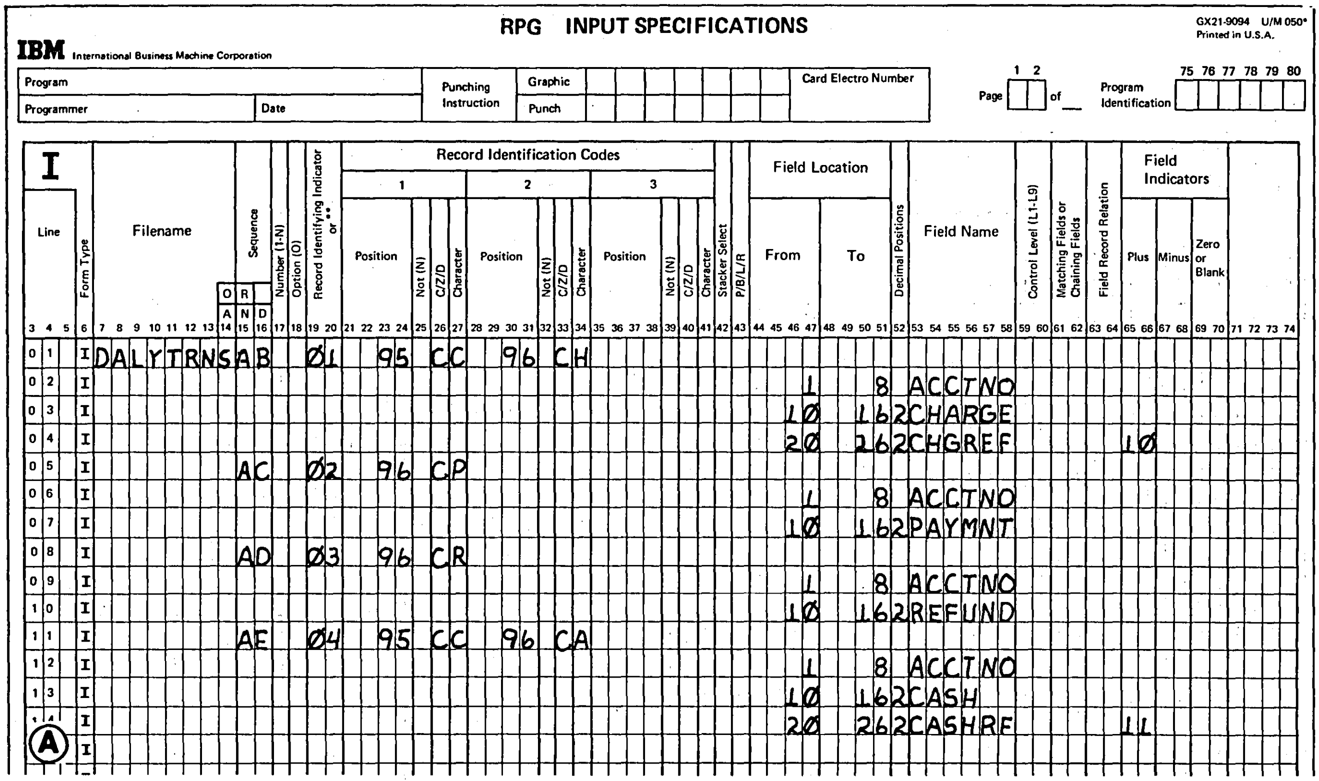
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

```
  01 C   Ø1                GOTO ROUT1
  02 C   Ø2                GOTO ROUT2
  03 C   Ø3                GOTO ROUT3
  04 C   Ø4                GOTO ROUT4
  05 C                     GOTO END
  06 C           ROUT1     TAG
  07 C
  08 C
  09 C
  10 C                     GOTO END
  11 C           ROUT2     TAG
  12 C
  13 C
  14 C                     GOTO END
  15 C           ROUT3     TAG
  16 C
  17 C
  18 C                     GOTO END
  19 C           ROUT4     TAG
  20 C
     C           END       TAG
     C
     C
     C
     C
```

Figure 5-44. Recommended Branching Structure



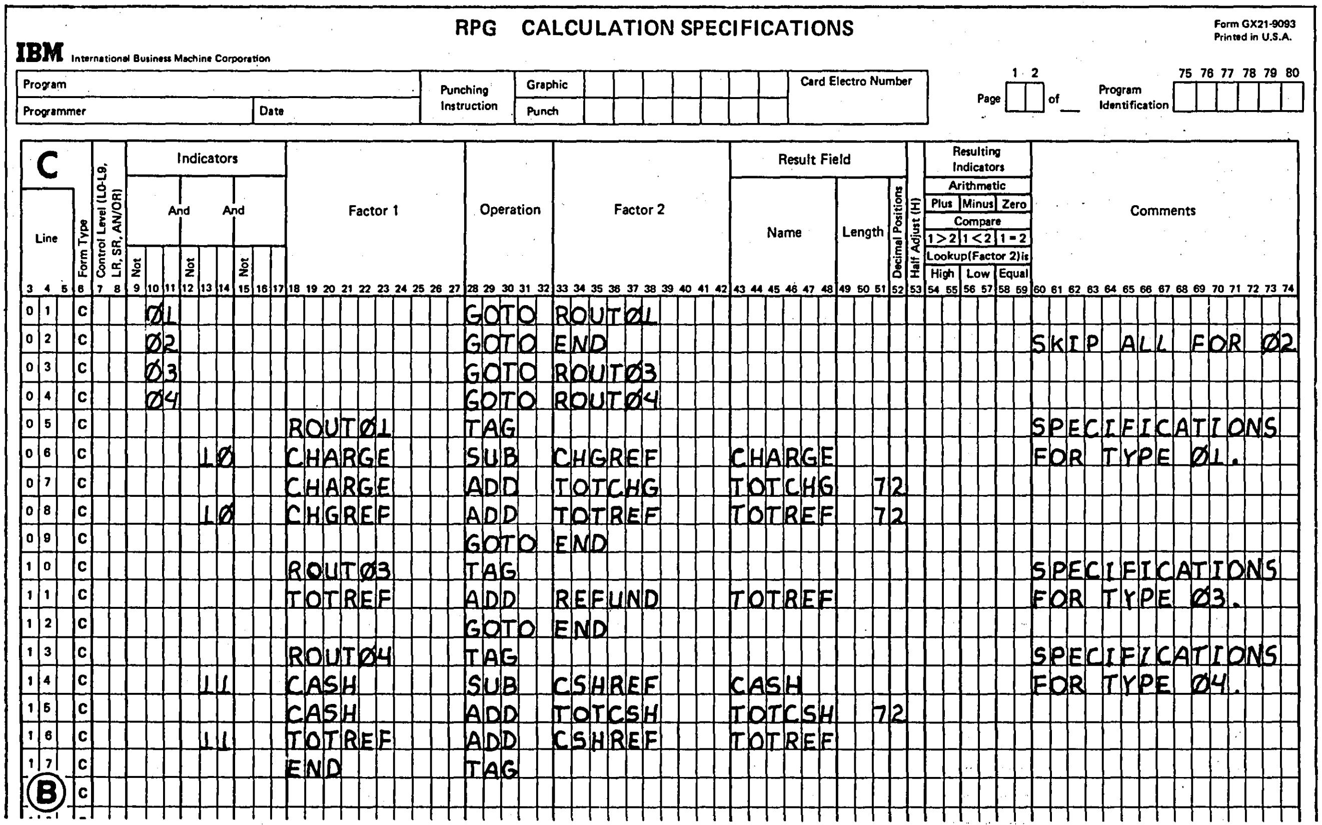
IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

```
  01 IDALYTRNSAB  Ø1  95 CC  96 CH
  02 I                                        1   8 ACCTNO
  03 I                                       1Ø  162CHARGE
  04 I                                       2Ø  262CHGREF      1Ø
  05 I        AC  Ø2  96 CP
  06 I                                        1   8 ACCTNO
  07 I                                       1Ø  162PAYMNT
  08 I        AD  Ø3  96 CR
  09 I                                        1   8 ACCTNO
  10 I                                       1Ø  162REFUND
  11 I        AE  Ø4  95 CC  96 CA
  12 I                                        1   8 ACCTNO
  13 I                                       1Ø  162CASH
  14 I                                       2Ø  262CASHRF      11
```

Ⓐ



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

```
  01 C   Ø1                GOTO ROUTØ1
  02 C   Ø2                GOTO END                        SKIP ALL FOR Ø2
  03 C   Ø3                GOTO ROUTØ3
  04 C   Ø4                GOTO ROUTØ4
  05 C           ROUTØ1    TAG                             SPECIFICATIONS
  06 C        1Ø CHARGE    SUB  CHGREF    CHARGE           FOR TYPE Ø1.
  07 C           CHARGE    ADD  TOTCHG    TOTCHG  72
  08 C        1Ø CHGREF    ADD  TOTREF    TOTREF  72
  09 C                     GOTO END
  10 C           ROUTØ3    TAG                             SPECIFICATIONS
  11 C           TOTREF    ADD  REFUND    TOTREF           FOR TYPE Ø3.
  12 C                     GOTO END
  13 C           ROUTØ4    TAG                             SPECIFICATIONS
  14 C        11 CASH      SUB  CSHREF    CASH             FOR TYPE Ø4.
  15 C           CASH      ADD  TOTCSH    TOTCSH  72
  16 C        11 TOTREF    ADD  CSHREF    TOTREF
  17 C           END       TAG
```

Ⓑ

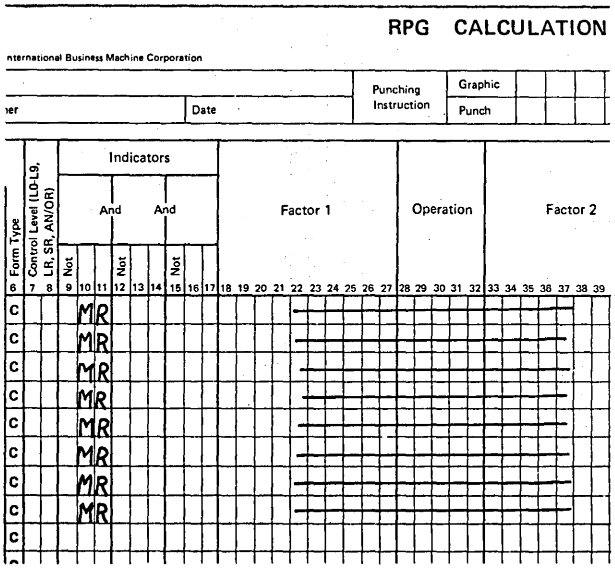
Figure 5-45. Sales Analysis Job

Figure 5-45, insert A, shows the input specifications for all record types. Notice the record identifying indicators assigned to each record type.

In the calculations, all charges, cash sales, and refunds are totaled. If a refund is given along with the cash or charge sales for an account, the amount of the refund is subtracted from the amount of the cash or charge before the cash or charge is added to the total. Figure 5-45, insert B, shows the calculation specifications for the job. As you see, the recommended branching structure was used. If a charge record (type 01) is read, a branch is taken to ROUT01. Calculations specified in lines 06-08 are performed. Then a branch is taken to END, for there are no more operations to do for that record. Branching for record types 03 and 04 is handled in the same way as for record 01. However, when a payment record is read (02 turns on) no calculations are performed because this job is not concerned with payments. Therefore, a branch is taken to END. In this way all calculations are bypassed. They are not even checked to see if conditions established by indicators are satisfied.

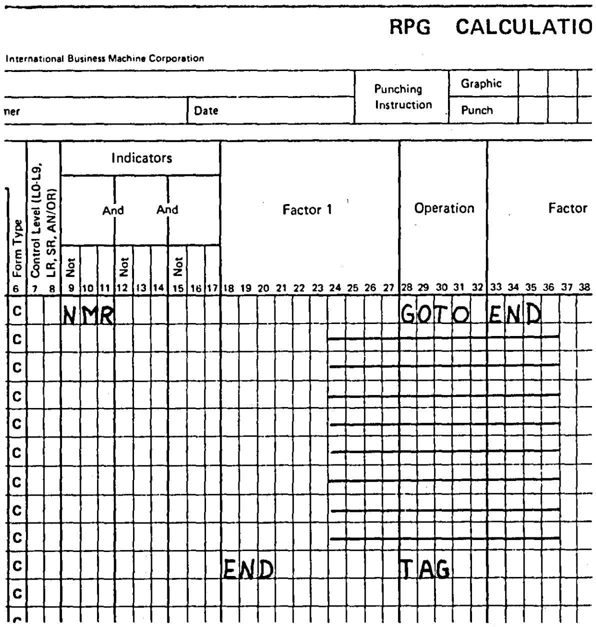
Branching in a Matching Records Job

Suppose that you are doing a matching record job which requires that all calculations be done when records match. All specifications can be conditioned by MR (see Figure 5-46), or GOTO and TAG can be used to branch around all calculations when records do not match (see Figure 5-47). In this example, GOTO is conditioned by NMR. When records do not match, all calculations are skipped and a branch to END occurs.


RPG CALCULATION
nternational Business Machine Corporation
ier
Date
Punching Instruction
Graphic
Punch
Form Type
Control Level (L0-L9, LR, SR, AN/OR)
Indicators
And
And
Not
Not
Not
Factor 1
Operation
Factor 2
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

```
C   MR
C   MR
C   MR
C   MR
C   MR
C   MR
C   MR
C   MR
C
```

Figure 5-46. Conditioning all Specifications by MR

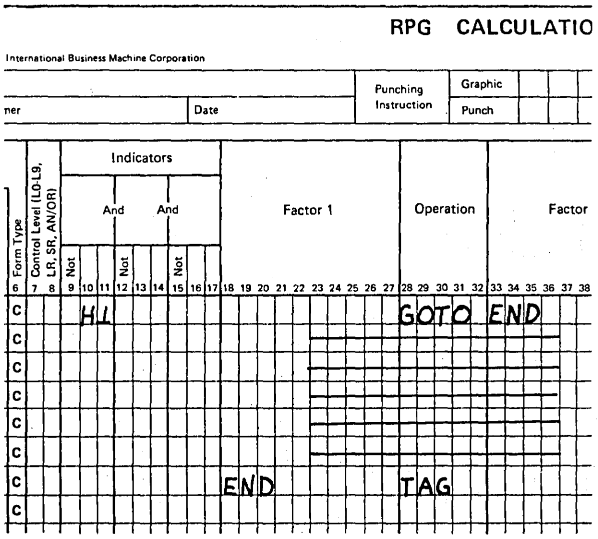

RPG CALCULATIO
International Business Machine Corporation
ner
Date
Punching Instruction
Graphic
Punch
Form Type
Control Level (L0-L9, LR, SR, AN/OR)
Indicators
And
And
Not
Not
Not
Factor 1
Operation
Factor
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38

```
C  NMR                GOTO END
C
C
C
C
C
C
C
C
C           END       TAG
C
```

Figure 5-47. Branching in a Matching Records Job

Branching When An Error Condition Occurs

Branching is an easy way to bypass all calculations which should not be performed when an error occurs. Figure 5-48 shows an example of this.


RPG CALCULATIO
International Business Machine Corporation
ner
Date
Punching Instruction
Graphic
Punch
Form Type
Control Level (L0-L9, LR, SR, AN/OR)
Indicators
And
And
Not
Not
Not
Factor 1
Operation
Factor
6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38

```
C   HL                GOTO END
C
C
C
C
C
C           END       TAG
C
```

Figure 5-48. Branching When an Error Occurs

Branching at Different Points in the Program

Within one program you may use any number of GOTO and TAG operation codes. One or more GOTO statements may have the same name (Figure 5-45, insert B, lines 02, 09, and 12). However, each TAG must have a different name. If two TAGs had the same name, your program would not know where to branch.

Branching at Detail and Total Time

You may branch from one detail calculation to another detail calculation or from one total calculation to another total calculation. However, you cannot branch from a detail to a total calculation or vice versa.

Branching Backward

So far, you have learned only about branching forward in order to skip statements that you do not wish to perform. The GOTO statement can also branch backward. In this way you can go back to statements that were already done in order to do them again.

Doing the same statements over and over again in one program cycle is called *looping.* The statements done several times in one cycle, plus the branching statement, make up the *loop.* Figure 5-49 shows the basic structure of a loop.

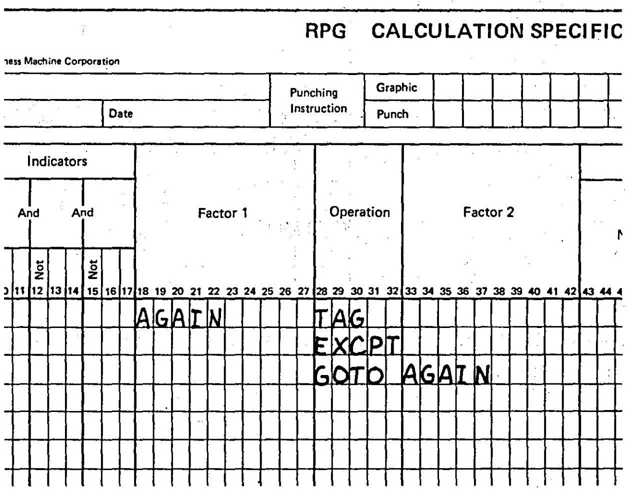
When would you want to branch backward? Suppose you want to print out several mailing labels for each customer by using the EXCPT operation code (see *Repetitive Output (EXCPT Operation)* in the chapter, *Programmed Control of Input and Output* for an explanation of the EXCPT operation). EXCPT is used to write several output records in one program cycle. If you want to put out 25 mailing labels for Joe Aaron, you would have to write the EXCPT operation 25 times – one time for each record. However, instead of writing EXCPT 25 times, you could use a loop. One EXCPT code is specified. It is performed and then a backward branch is taken so that EXCPT will be done again.

Figure 5-49. Structure of a Loop

Look at Figure 5-50. The loop just described is coded here. But look at what will happen: An EXCPT record will be printed; a backward branch will cause the EXCPT to be done again; then another branch, and another EXCPT. When would execution of the statements in the loop end? As coded here, execution of these statements would go on indefinitely.

You want to stop printing mailing labels for Joe Aaron after 25 have been made. Therefore, you have to keep track of the number printed. This is done through the use of a special count field which is constantly updated to reflect the number of times looping has occurred. Figure 5-51 shows the way this is done. At the beginning of each cycle, COUNT, the field used to keep track of the number of records printed, must be zero. Each time a record is put out, 1 is added to COUNT. COUNT is then compared to 25, the number of mailing labels desired. When COUNT equals 25, calculations will be complete for that program cycle, and looping will stop. When COUNT is less than 25 (indicator 10 is on), a branch is taken back to the beginning of the calculations so that they can be done again.

In this example, 25 mailing labels are created each cycle (for each record read). If a *different* number of records are required to be printed or punched for each cycle, you have to compare the COUNT field to another *field* which contains the number of records to be put out in that cycle (see Figure 5-52).



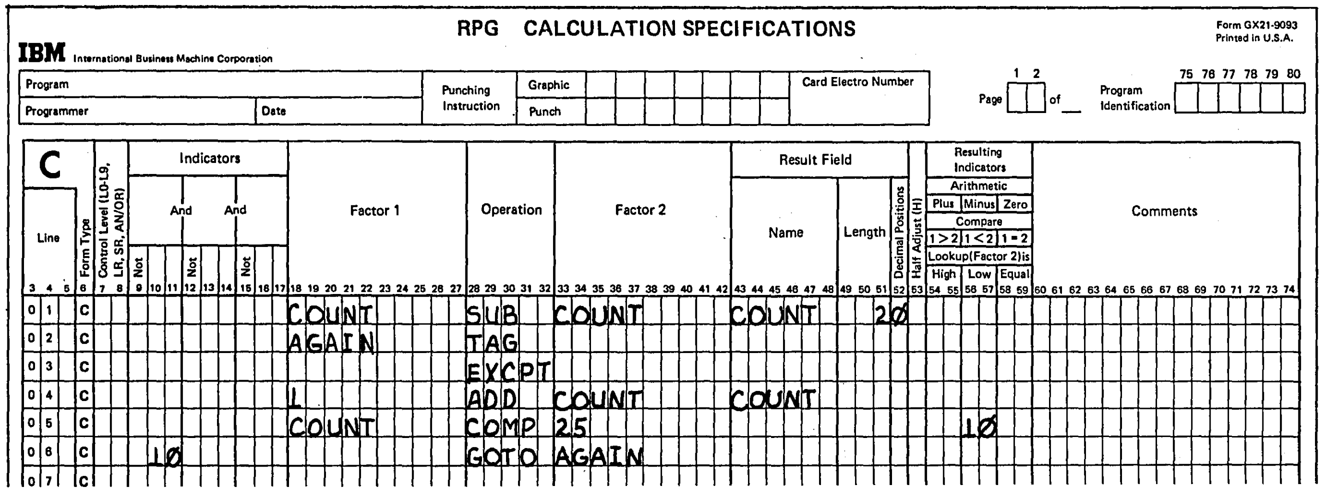
RPG CALCULATION SPECIFIC

ness Machine Corporation

Date	Punching Instruction	Graphic	
		Punch	

Indicators (And, Not, And, Not)	Factor 1 (18–27)	Operation (28–32)	Factor 2 (33–42)
	AGAIN	TAG	
		EXCPT	
		GOTO	AGAIN

Figure 5-50. An Uncontrolled Loop



IBM International Business Machine Corporation

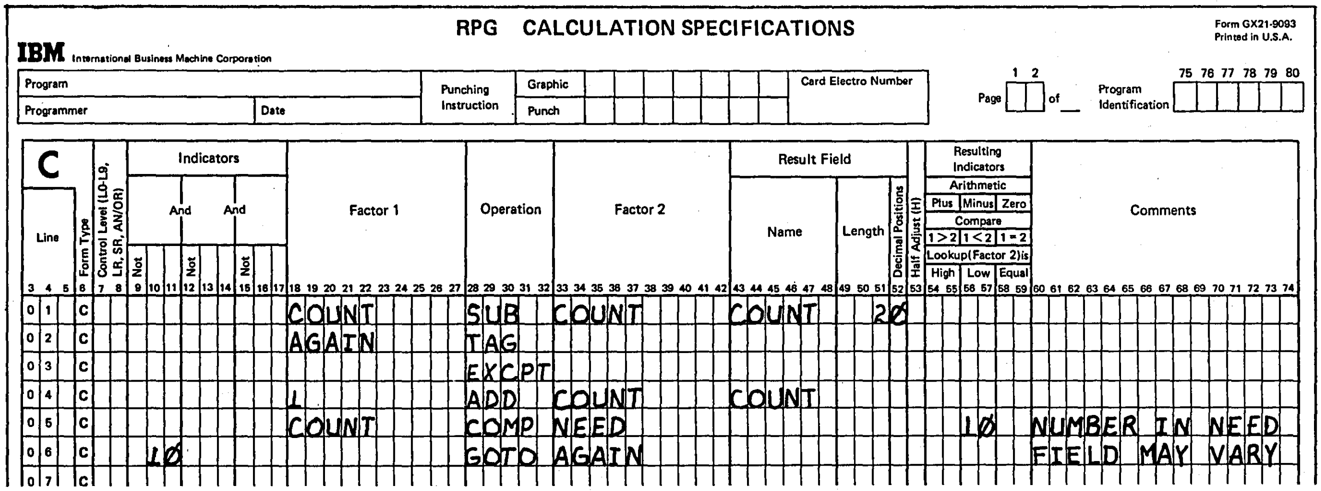
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	Resulting Indicators High	Low	Equal	Comments
01	C			COUNT	SUB	COUNT	COUNT	2	Ø					
02	C			AGAIN	TAG									
03	C				EXCPT									
04	C			1	ADD	COUNT	COUNT							
05	C			COUNT	COMP	25						1Ø		
06	C		1Ø		GOTO	AGAIN								
07	C													

Figure 5-51. A Controlled Loop



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	Resulting Indicators High	Low	Equal	Comments
01	C			COUNT	SUB	COUNT	COUNT	2	Ø					
02	C			AGAIN	TAG									
03	C				EXCPT									
04	C			1	ADD	COUNT	COUNT							
05	C			COUNT	COMP	NEED						1Ø		NUMBER IN NEED
06	C		1Ø		GOTO	AGAIN								FIELD MAY VARY
07	C													

Figure 5-52. Printing a Various Number of Records in Each Cycle

You can have as many loops in your program as you need. You can even have one loop within another loop (see Figure 5-53, insert A).

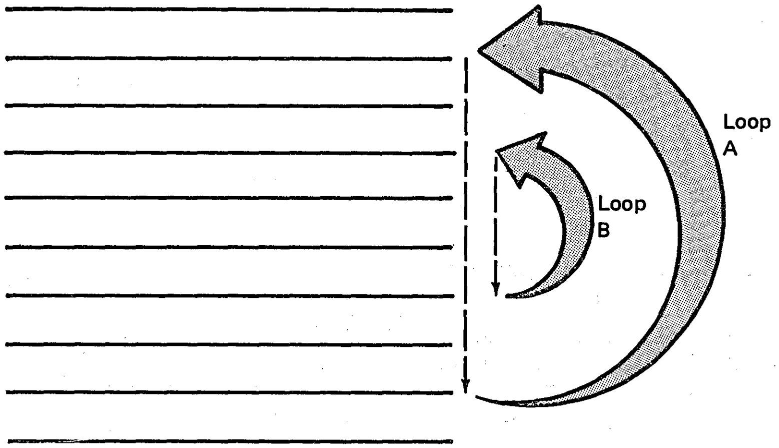
Keep in mind, however, that an uncontrolled loop is an endless loop. Each loop you create must be controlled so that it will be performed only a limited number of times. Always set up a condition which when satisfied will cause looping to stop (see Figure 5-53, insert B).

USING SUBROUTINES IN CALCULATIONS

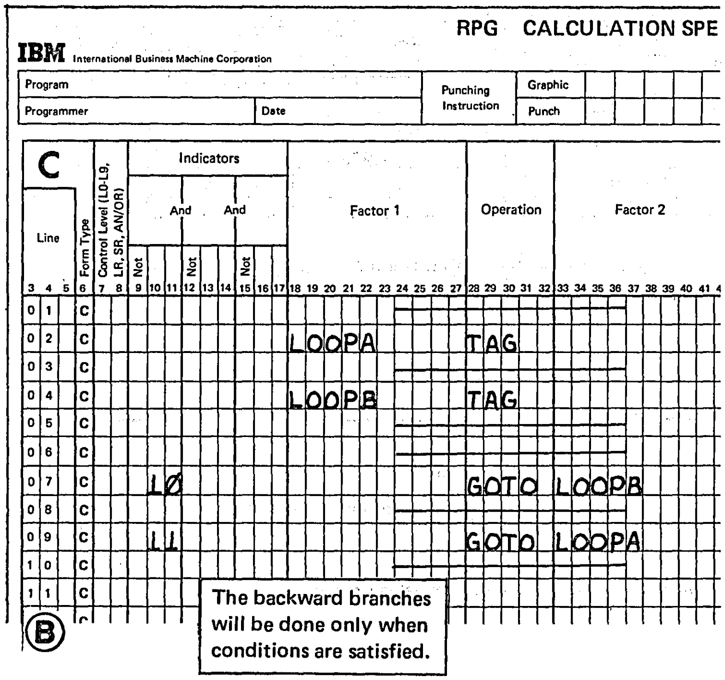
A routine is something done over and over again. The calculations in your program can be called a routine because the operations are done again and again (on each program cycle). A *subroutine* is a routine that is part of another routine. That is, a subroutine is a group of operations that can be used by the main routine several times in the same program cycle. A subroutine can also be a sequence of operations that is coded once and included in several different programs.

Subroutines can be used to:

- Eliminate duplicate coding by performing the same calculations several times in the same program cycle or in several different programs.
- Reduce the storage requirements of RPG II programs (see *Controlling Overlay by Using Subroutines* in this chapter).



Ⓐ



RPG CALCULATION SPE

IBM International Business Machine Corporation

Program			Punching Instruction	Graphic	
Programmer		Date		Punch	

C

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2
01	C					
02	C			LOOPA	TAG	
03	C					
04	C			LOOPB	TAG	
05	C					
06	C					
07	C		LØ		GOTO	LOOPB
08	C					
09	C		L1		GOTO	LOOPA
10	C					
11	C					
	C					

The backward branches will be done only when conditions are satisfied.

Ⓑ

Figure 5-53. A Loop Within a Loop

Using Subroutines to do the Same Calculations Several Times in One Cycle

In many of your programs, the same operation(s) may be required several times in one cycle: When coding the job, you can specify the operations as many times as needed. This often involves large amounts of coding, however. If the same operations are to be done several times in succession, you can often use looping to reduce the amount of coding. This was explained in the previous section.

What if the same operations are not done several times in succession, but are performed, instead, at many different points in your program? Creating a loop couldn't work in this case. As an example, consider the job which creates a weekly sales commission report. The report desired (Figure 5-54) shows two things:

1. Total commission earned by each salesman.

2. Total commissions paid in each district.

The area in which all salesmen work is divided into three districts: A, B, and C. Some salesmen work in only one district. Others may work in parts of two or more districts.

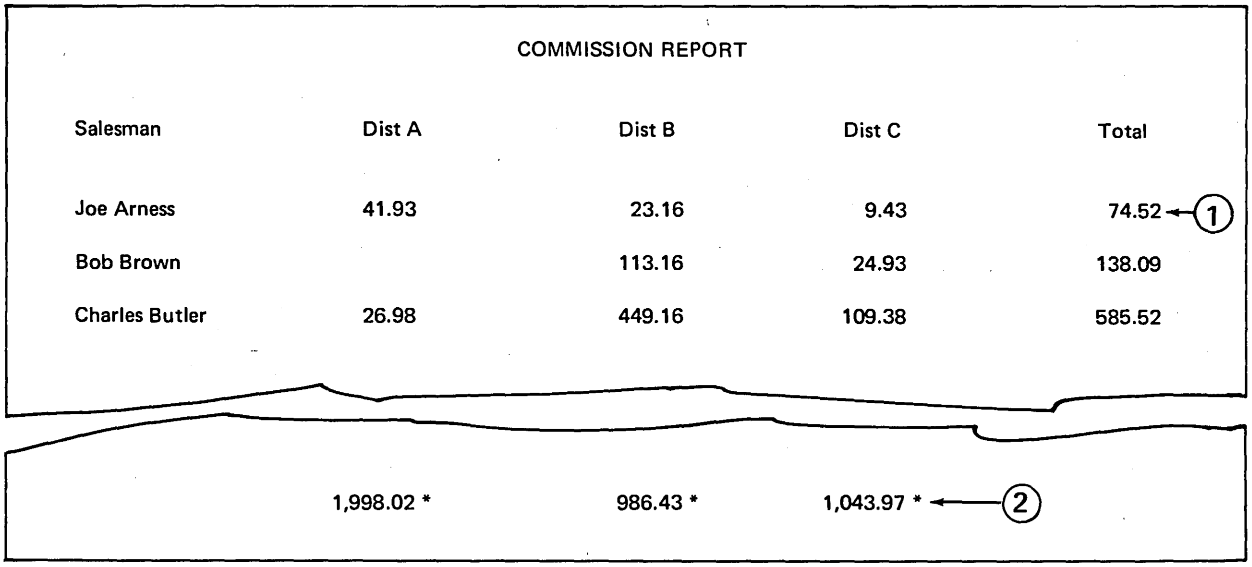
For each salesman, the input file contains a record formatted as shown in Figure 5-55. The amounts in the district fields show total weekly sales made by that salesman in each district. If the salesman did not work a district or made no sales in that district, the field contains a zero.

The report must contain the commission earned in each district by each salesman. In addition, total commission must be accumulated for each salesman and for each district. The percentage of commission is:

- 3 percent of the gross sales .01 to 1000.00 dollars.

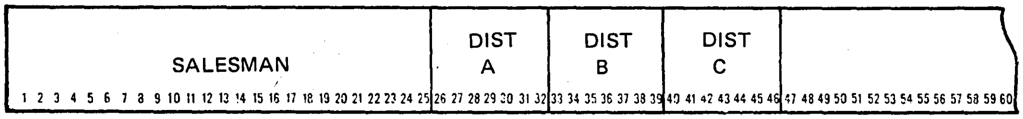
- Plus 2 percent of the gross sales 1000.01 to 5000.00 dollars.

- Plus 1 percent of the gross over 5000.00 dollars.



```
                                COMMISSION REPORT

Salesman            Dist A          Dist B          Dist C          Total

Joe Arness          41.93            23.16            9.43           74.52
Bob Brown                           113.16           24.93          138.09
Charles Butler      26.98           449.16          109.38          585.52


                 1,998.02 *        986.43 *      1,043.97 *
```

Figure 5-54. Sales Commission Report



SALESMAN	DIST A	DIST B	DIST C	
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25	26 27 28 29 30 31 32	33 34 35 36 37 38 39	40 41 42 43 44 45 46	47 48 49 50 51 52 53 54 55 56 57 58 59 60

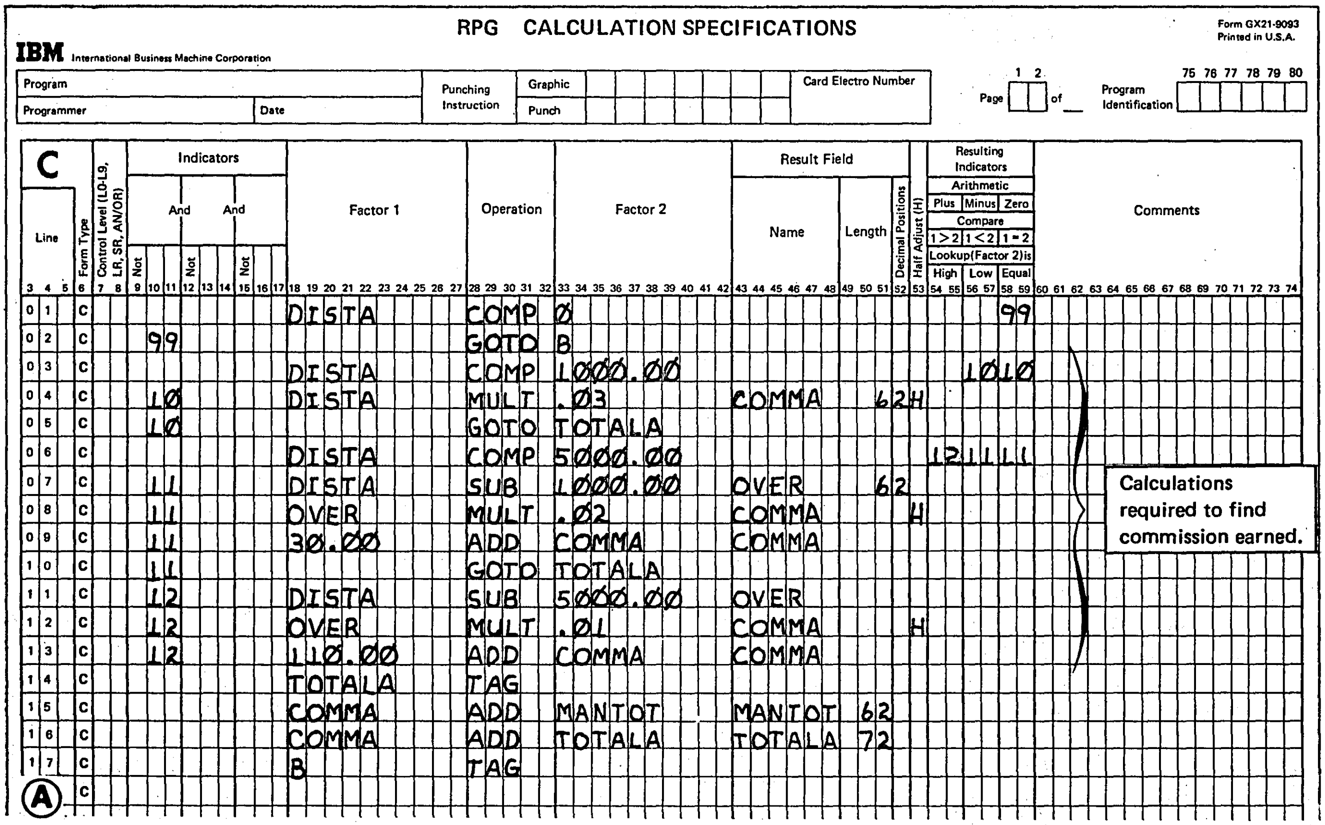
Figure 5-55. Input Record for Sales Commission Report

Figure 5-56 shows the calculations needed to find the information required for the report. You first compare the contents of each district field to zero to find out if the salesman sold anything in that district. If it is not zero, you calculate the commission (COMM) earned. You then add commission earned to total commission for the salesman (MANTOT) and to total commission paid in each district (TOTALA, TOTALB, or TOTALC).

The calculations needed to find commission earned are the same for each district (Figure 5-56, inserts A, B, C, lines 3-16). Why code these calculations three times? Why not code them once and branch to them each time they are needed? (See Figure 5-57.)

Using the branching you have learned about so far, the branching that uses GOTO and TAG, you could easily branch to the calculations needed to find commission. But since you could branch to them from *three* different places, it would be difficult to determine to which point in the calculations you should return. You could return to the point where totals are accumulated for district A, the point they are accumulated for district B, or the point they are accumulated for district C. Imagine the number of indicators and SETON and SETOF operations necessary to do this. Wouldn't it be nice if the program would automatically go back to the point from which it branched?

RPG II can do this through the use of a subroutine. Your entire program is the main routine. The operations to be performed several times in the program cycle make up a subroutine. In this case, the main program uses the subroutine to find commission earned.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

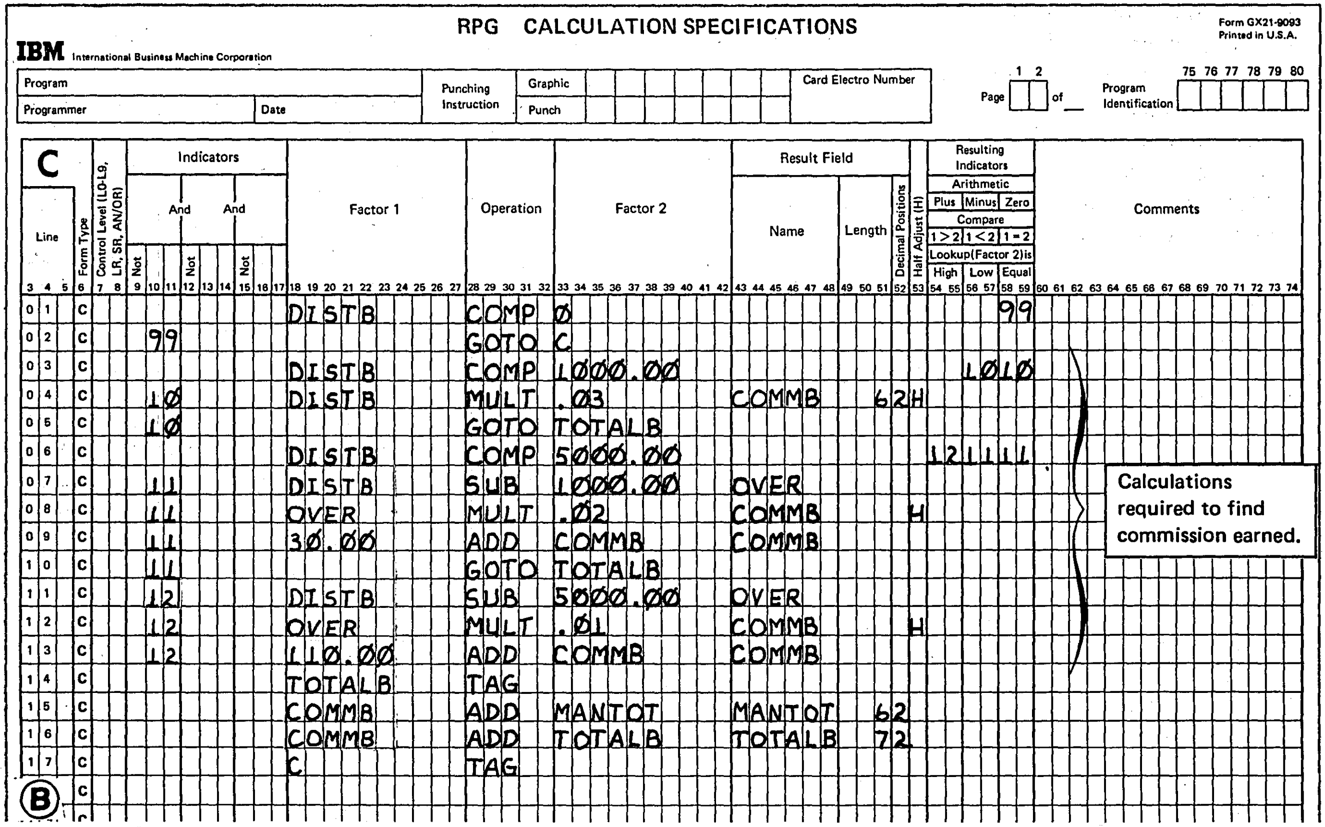
Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

C

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
01	C			DISTA	COMP	Ø							99	
02	C		99		GOTO	B								
03	C			DISTA	COMP	1ØØØ.ØØ						1Ø	1Ø	
04	C		1Ø	DISTA	MULT	.Ø3	COMMA	6	2	H				
05	C		1Ø		GOTO	TOTALA								
06	C			DISTA	COMP	5ØØØ.ØØ					12	11	11	
07	C		11	DISTA	SUB	1ØØØ.ØØ	OVER	6	2					Calculations required to find commission earned.
08	C		11	OVER	MULT	.Ø2	COMMA			H				
09	C		11	3Ø.ØØ	ADD	COMMA	COMMA							
10	C		11		GOTO	TOTALA								
11	C		12	DISTA	SUB	5ØØØ.ØØ	OVER							
12	C		12	OVER	MULT	.Ø1	COMMA			H				
13	C		12	11Ø.ØØ	ADD	COMMA	COMMA							
14	C			TOTALA	TAG									
15	C			COMMA	ADD	MANTOT	MANTOT	6	2					
16	C			COMMA	ADD	TOTALA	TOTALA	7	2					
17	C			B	TAG									
Ⓐ	C													

Figure 5-56 (Part 1 of 2). Calculations for Sales Commission Program



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

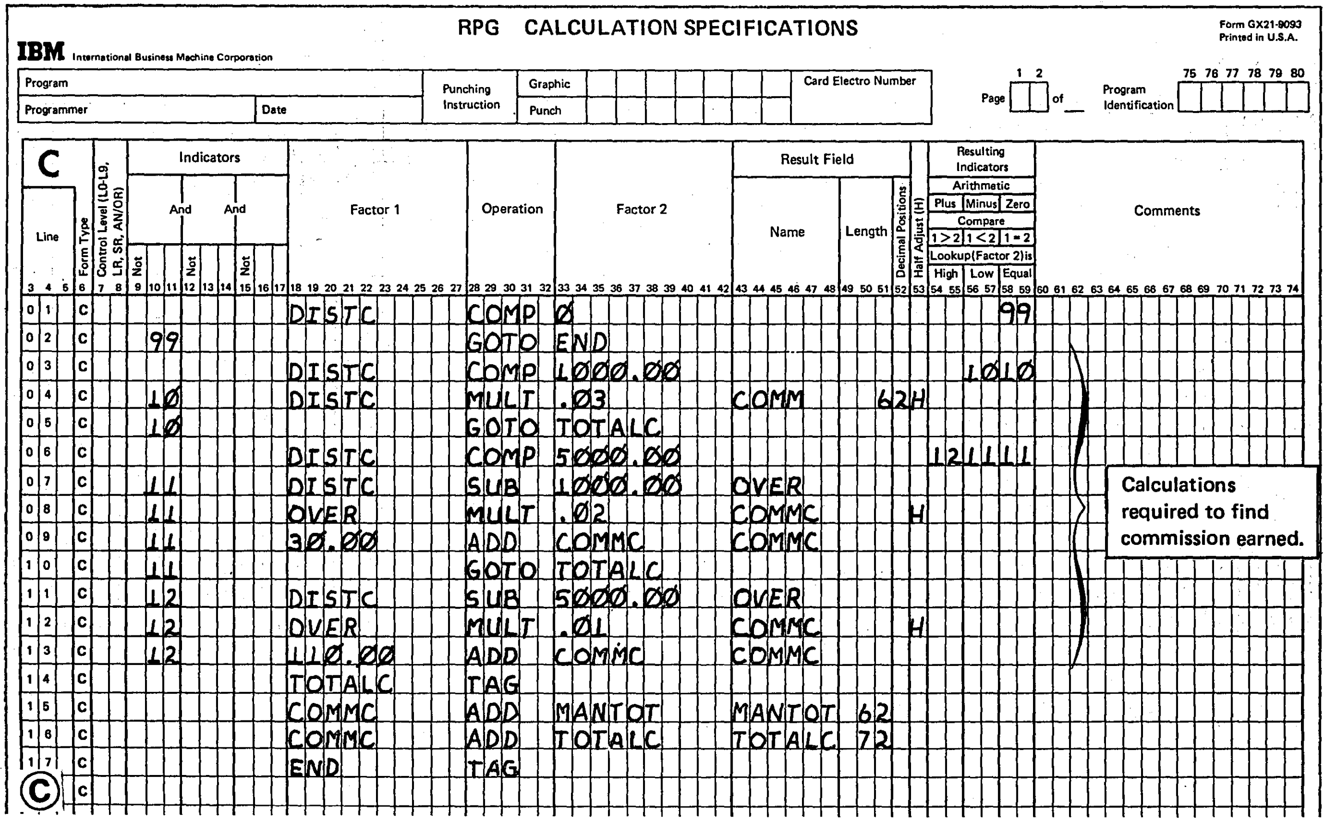
Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	High	Low	Equal	Comments
01	C		DISTB	COMP	Ø							99	
02	C	99		GOTO	C								
03	C		DISTB	COMP	1ØØØ.ØØ						1Ø	1Ø	
04	C	1Ø	DISTB	MULT	.Ø3	COMMB	6	2	H				
05	C	1Ø		GOTO	TOTALB								
06	C		DISTB	COMP	5ØØØ.ØØ					12	11	11	
07	C	11	DISTB	SUB	1ØØØ.ØØ	OVER							
08	C	11	OVER	MULT	.Ø2	COMMB			H				
09	C	11	3Ø.ØØ	ADD	COMMB	COMMB							
10	C	11		GOTO	TOTALB								
11	C	12	DISTB	SUB	5ØØØ.ØØ	OVER							
12	C	12	OVER	MULT	.Ø1	COMMB			H				
13	C	12	11Ø.ØØ	ADD	COMMB	COMMB							
14	C		TOTALB	TAG									
15	C		COMMB	ADD	MANTOT	MANTOT	6	2					
16	C		COMMB	ADD	TOTALB	TOTALB	7	2					
17	C		C	TAG									
	C												

Calculations required to find commission earned.

B



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	High	Low	Equal	Comments
01	C		DISTC	COMP	Ø							99	
02	C	99		GOTO	END								
03	C		DISTC	COMP	1ØØØ.ØØ						1Ø	1Ø	
04	C	1Ø	DISTC	MULT	.Ø3	COMM	6	2	H				
05	C	1Ø		GOTO	TOTALC								
06	C		DISTC	COMP	5ØØØ.ØØ					12	11	11	
07	C	11	DISTC	SUB	1ØØØ.ØØ	OVER							
08	C	11	OVER	MULT	.Ø2	COMMC			H				
09	C	11	3Ø.ØØ	ADD	COMMC	COMMC							
10	C	11		GOTO	TOTALC								
11	C	12	DISTC	SUB	5ØØØ.ØØ	OVER							
12	C	12	OVER	MULT	.Ø1	COMMC			H				
13	C	12	11Ø.ØØ	ADD	COMMC	COMMC							
14	C		TOTALC	TAG									
15	C		COMMC	ADD	MANTOT	MANTOT	6	2					
16	C		COMMC	ADD	TOTALC	TOTALC	7	2					
17	C		END	TAG									
	C												

Calculations required to find commission earned.

C

Figure 5-56 (Part 2 of 2). Calculations for Sales Commission Program

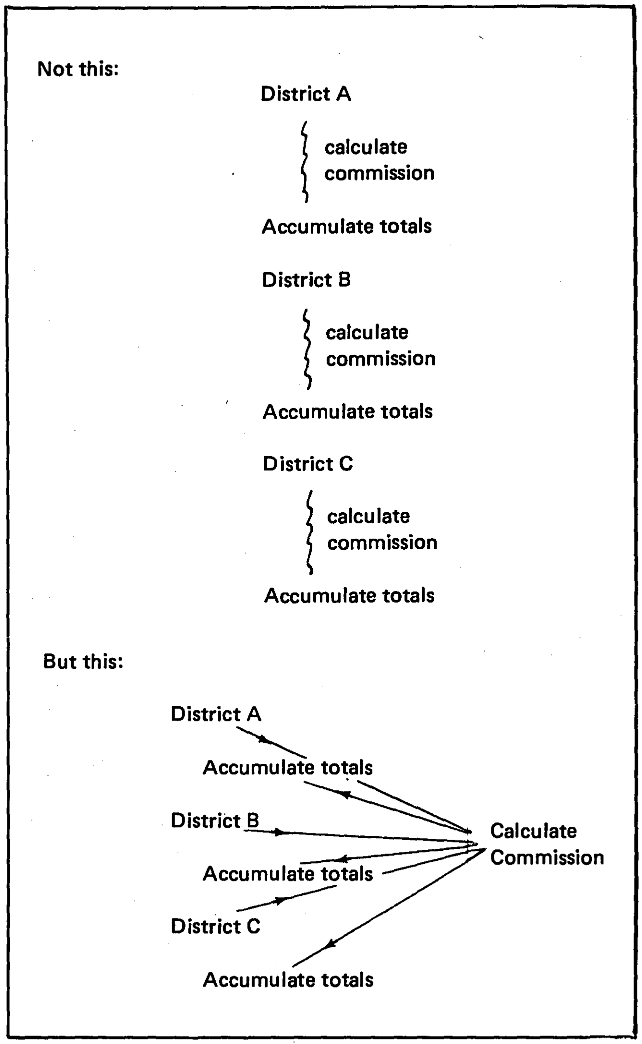
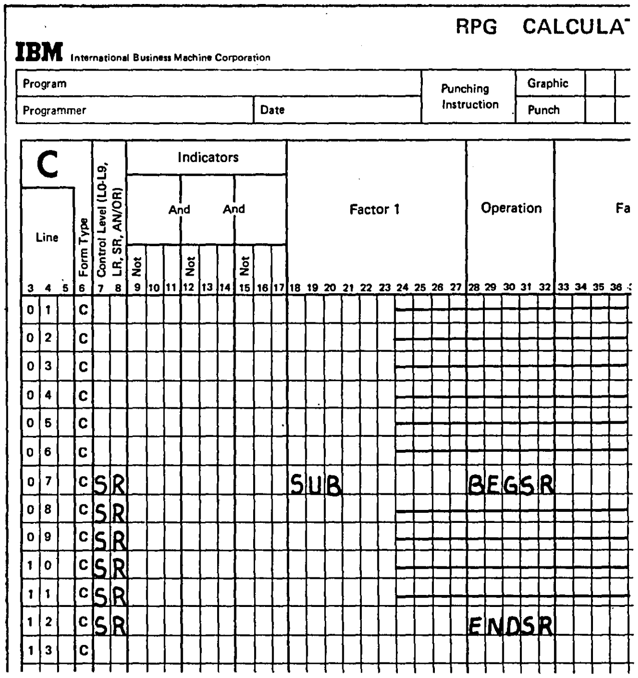


Figure 5-57. Branching to Similar Calculations

Specifications for Coding a Subroutine

Subroutines are specified on the Calculation sheet after all detail and total operations. Every statement in the subroutine must be identified as part of the subroutine by the letters SR in columns 7-8. (See Figure 5-58.) In addition, two operation codes, BEGSR and ENDSR, are needed to establish the beginning and end of the subroutine.

Every subroutine used in the program must have a unique name. The rules for forming a subroutine name are the same as those for forming a field name. The name must appear in Factor 1 on the same line as the BEGSR operation code.



IBM International Business Machine Corporation

RPG CALCULA

Program

Programmer

Date

Punching Instruction

Graphic

Punch

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Fa
01	C					
02	C					
03	C					
04	C					
05	C					
06	C					
07	C	SR		SUB	BEGSR	
08	C	SR				
09	C	SR				
10	C	SR				
11	C	SR				
12	C	SR			ENDSR	
13	C					

Figure 5-58. Structure of a Subroutine

Calling the Subroutine

When using GOTO and TAG, you use a GOTO operation code to branch to the next operation to be performed. When you do the operations in a subroutine, you do not branch to the subroutine; you *call* it.

When you call a subroutine, you use the execute subroutine (EXSR) operation code. This operation code can be placed anywhere in the calculation operations. Whenever the EXSR operation code is encountered, all operations in the subroutine will be performed. After the subroutine has been executed, RPG II branches back to the main program and continues execution with the next statement after the EXSR statement (Figure 5-59).

Factor 2 must contain the name of the subroutine to be executed. This same name must appear on a BEGSR instruction.

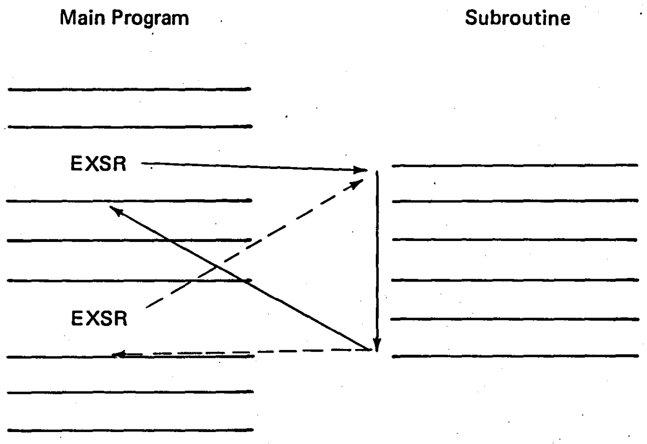


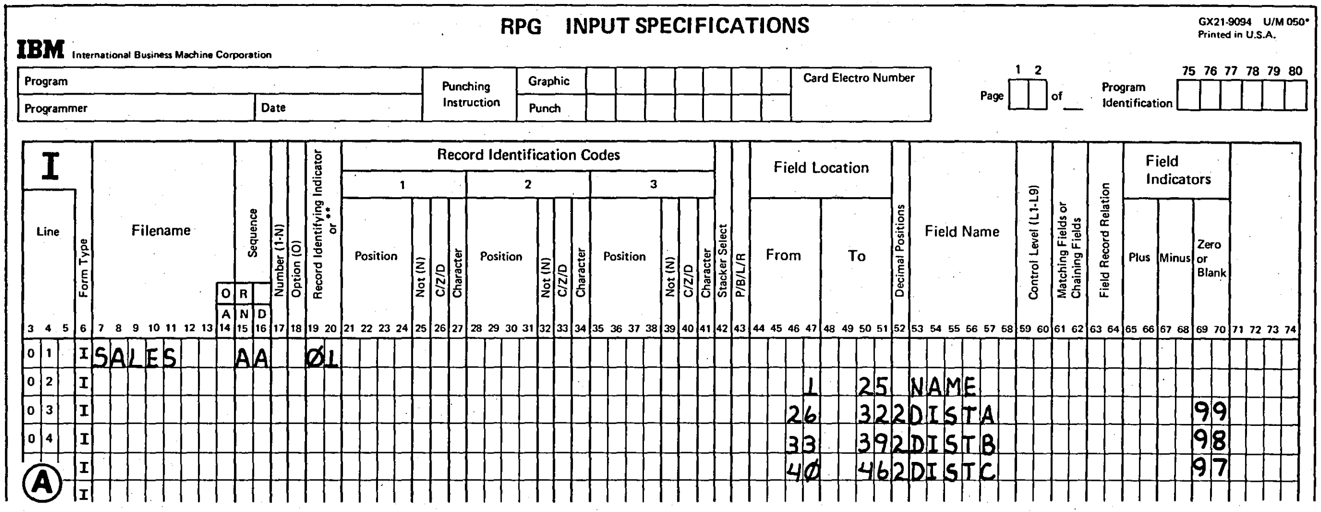
Figure 5-59. EXSR (Order in Which Calculations are Performed)

Fields Used in a Subroutine

The same fields can be used by both the subroutine and the main routine. You may define the field in either routine. However, the name and characteristics of the field must be the same in both routines.

The fields you define in a subroutine should be general so that they apply to all situations for which a subroutine is used. For example, if DISTA is used as the field name in a subroutine to calculate district sales, you *always* take information from the DISTA field when calculating commission. However, you want the routine also to handle information from the fields DISTB and DISTC. Using specific fields limits the correct use of a subroutine to one situation.

Instead, if you use a general field name such as SALES, this one subroutine can be used to calculate commission in all three districts (Figure 5-60, insert C). However, because there is no input field called SALES, you should use the Z-ADD operation code to place information in this field (Figure 5-60, insert B). The information in the appropriate district field (DISTA, DISTB, or DISTC) is moved into the field called SALES before the subroutine is executed. When finding commission earned in district A, DISTA is moved into SALES; when finding commission earned for district B, DISTB is moved into SALES, etc. In this way, you ensure that the subroutine uses the correct information each time it is called.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

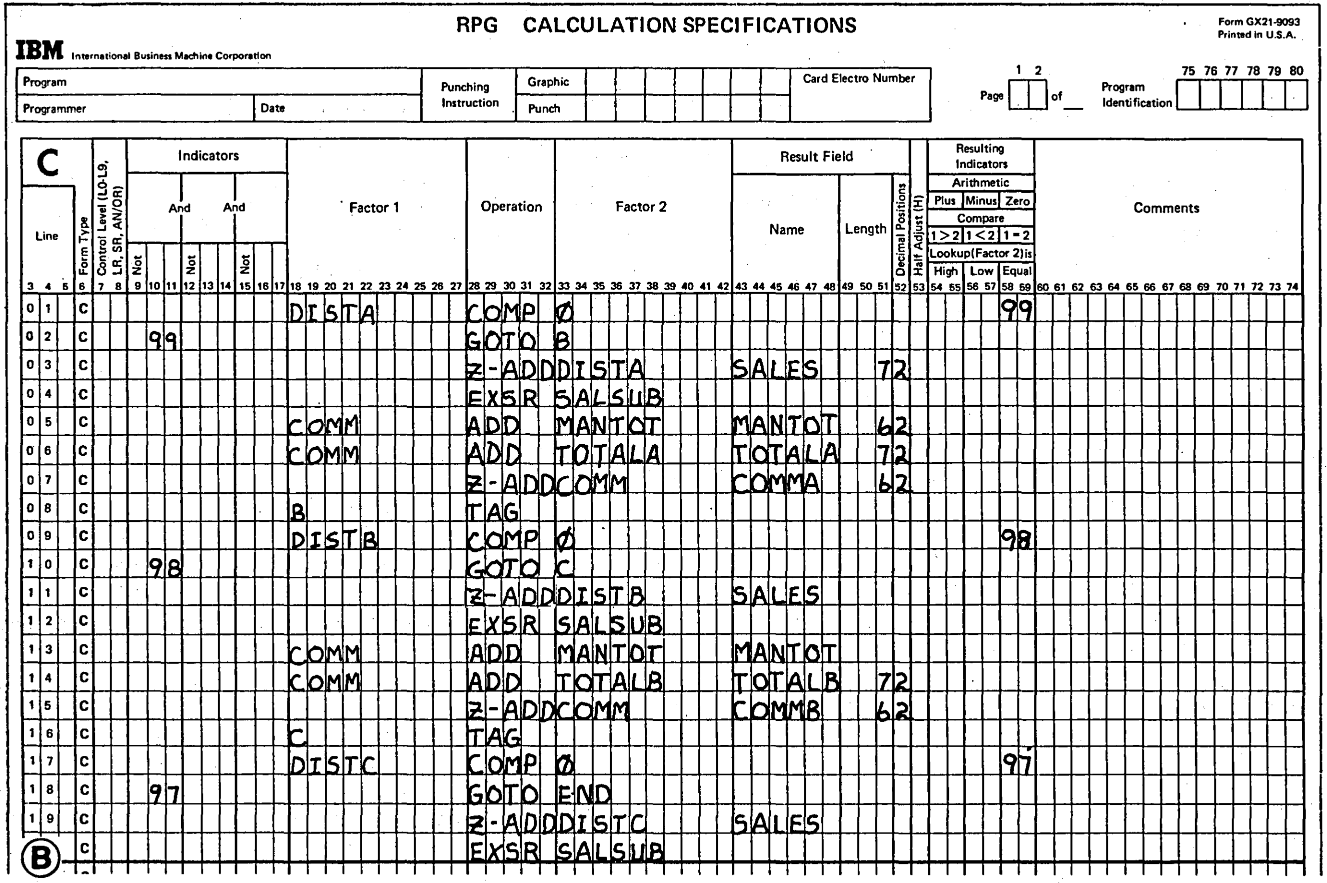
GX21-9094 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction — Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

I

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	1 Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2 Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3 Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P/B/L/R (43)	From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	(71-74)
01	I	SALES	AA			Ø1																									
02	I																				1	25		NAME							
03	I																				26	32	2	DISTA						99	
04	I																				33	39	2	DISTB						98	
A	I																				4Ø	46	2	DISTC						97	
	I																														

Figure 5-60 (Part 1 of 4). Sales Commission Program (Using a Subroutine)



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page __ of __ | Program Identification

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Resulting Indicators High	Low	Equal
01	C		DISTA	COMP	Ø						99
02	C	99		GOTO	B						
03	C			Z-ADD	DISTA	SALES	7	2			
04	C			EXSR	SALSUB						
05	C		COMM	ADD	MANTOT	MANTOT	6	2			
06	C		COMM	ADD	TOTALA	TOTALA	7	2			
07	C			Z-ADD	COMM	COMMA	6	2			
08	C		B	TAG							
09	C		DISTB	COMP	Ø						98
10	C	98		GOTO	C						
11	C			Z-ADD	DISTB	SALES					
12	C			EXSR	SALSUB						
13	C		COMM	ADD	MANTOT	MANTOT					
14	C		COMM	ADD	TOTALB	TOTALB	7	2			
15	C			Z-ADD	COMM	COMMB	6	2			
16	C		C	TAG							
17	C		DISTC	COMP	Ø						97
18	C	97		GOTO	END						
19	C			Z-ADD	DISTC	SALES					
Ⓑ	C			EXSR	SALSUB						

Figure 5-60 (Part 2 of 4). Sales Commission Program (Using a Subroutine)

Using Subroutines in the Sales Commission Report Example

Now that you have learned how subroutines are used, defined, and executed, see how they are used in the sales commission report program. All specifications are shown in Figure 5-60.

First a record is read. Now commission earned in each district must be calculated.

1. DISTA is compared to zero to see if the salesman sold anything in that district. If the field is greater than zero, commission must be calculated. If the field is zero, a branch is taken to B, where another comparison is made.

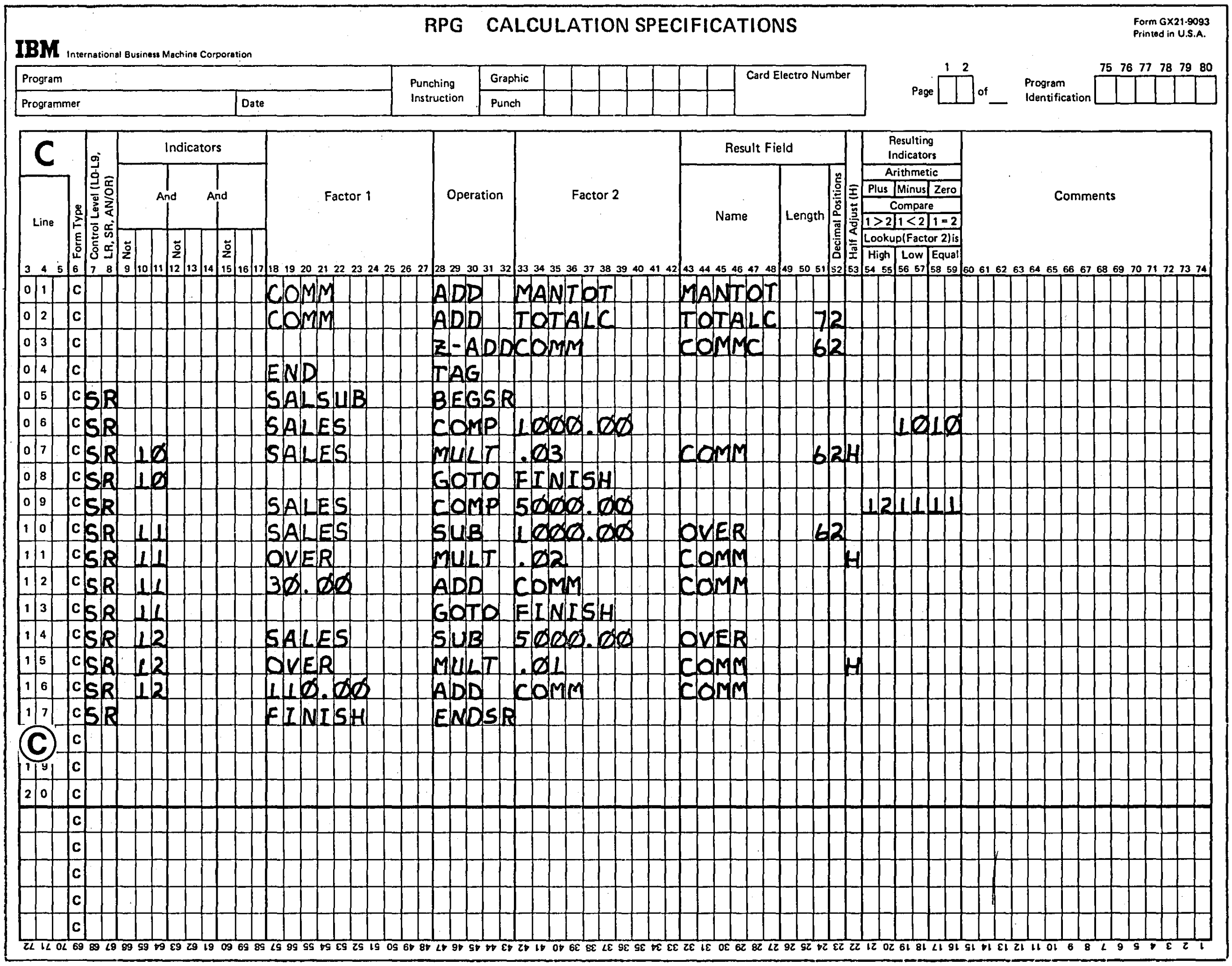
2. Before the subroutine can be called, it must be supplied with the correct amount of sales. Thus, the contents of DISTA are moved into SALES.

3. The subroutine is called by the EXSR operation code.

4. The commission is calculated by operations specified in the subroutine.

5. The subroutine is finished when ENDSR statement is executed. The instruction following EXSR is executed. The commission found by the subroutine is added to the total commission earned by the salesman (MANTOT) and to the total commission paid in the district (TOTALA).

6. Now DISTB is compared to zero to see if commission earned should be calculated. If so, information from the field DISTB is moved to SALES, and the subroutine is called. The next steps are basically the same as those already described. Follow the rest of the program.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

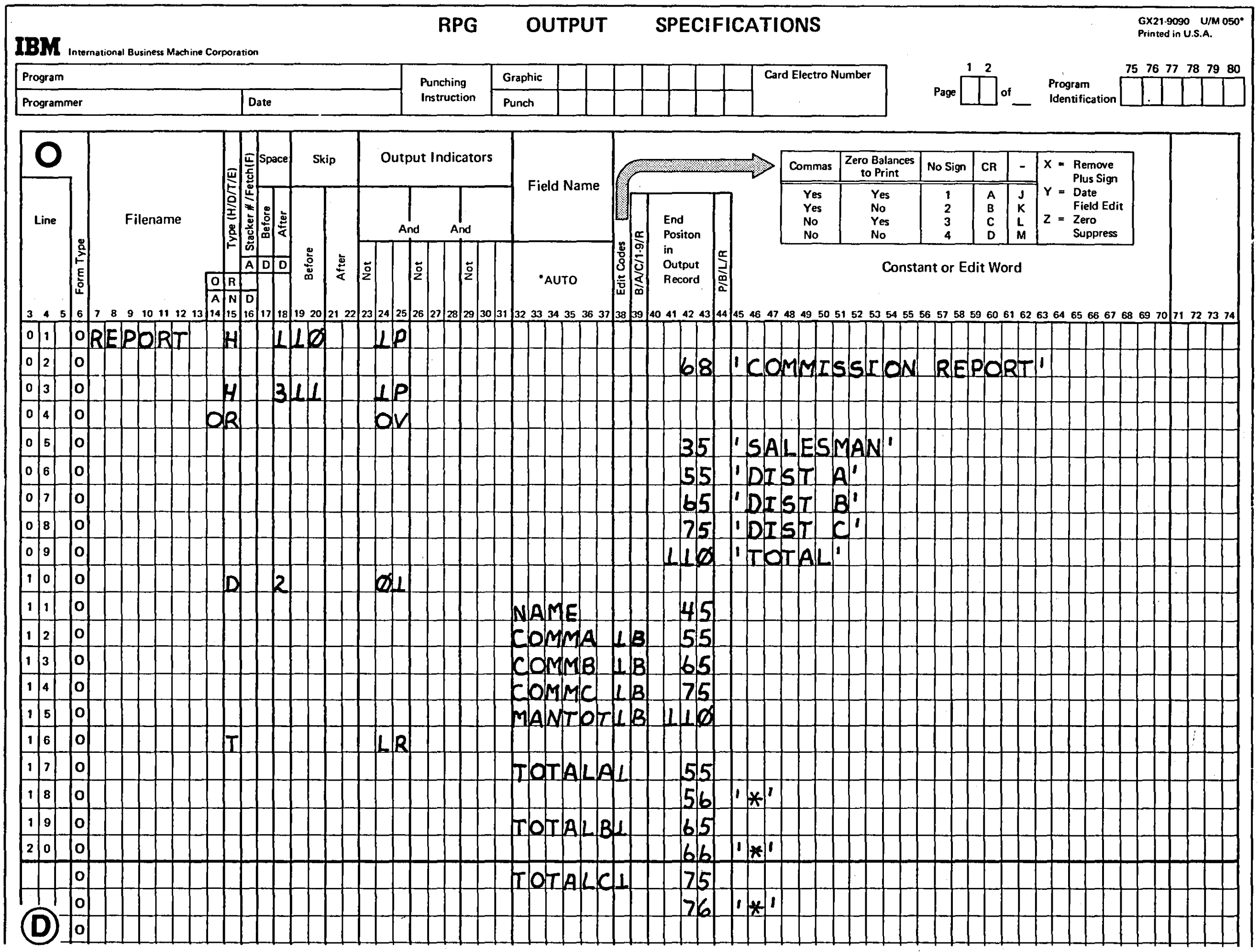
Form GX21-9093
Printed in U.S.A.

Program	
Programmer	Date
Punching Instruction	Graphic / Punch
Card Electro Number	
Page 1 2 of	Program Identification 75 76 77 78 79 80

C

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not (9-11)	And: Not (12-14)	And: Not (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field: Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare 1>2 / Lookup(Factor 2)is High (54-55)	Minus / 1<2 / Low (56-57)	Zero / 1=2 / Equal (58-59)	Comments (60-74)
01	C					COMM	ADD	MANTOT	MANTOT							
02	C					COMM	ADD	TOTALC	TOTALC	7	2					
03	C						Z-ADD	COMM	COMMC	6	2					
04	C					END	TAG									
05	C	SR				SALSUB	BEGSR									
06	C	SR				SALES	COMP	1ØØØ.ØØ						1Ø	1Ø	
07	C	SR	1Ø			SALES	MULT	.Ø3	COMM	6	2	H				
08	C	SR	1Ø				GOTO	FINISH								
09	C	SR				SALES	COMP	5ØØØ.ØØ					12	11	11	
10	C	SR	11			SALES	SUB	1ØØØ.ØØ	OVER	6	2					
11	C	SR	11			OVER	MULT	.Ø2	COMM			H				
12	C	SR	11			3Ø.ØØ	ADD	COMM	COMM							
13	C	SR	11				GOTO	FINISH								
14	C	SR	12			SALES	SUB	5ØØØ.ØØ	OVER							
15	C	SR	12			OVER	MULT	.Ø1	COMM			H				
16	C	SR	12			11Ø.ØØ	ADD	COMM	COMM							
17	C	SR				FINISH	ENDSR									
©	C															
19	C															
20	C															
	C															
	C															
	C															
	C															
	C															

Figure 5-60 (Part 3 of 4). Sales Commission Program (Using a Subroutine)



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Programmer | Date

Commas	Zero Balances to Print	No Sign	CR	-	X = Remove Plus Sign Y = Date Field Edit Z = Zero Suppress
Yes	Yes	1	A	J	
Yes	No	2	B	K	
No	Yes	3	C	L	
No	No	4	D	M	

```
01 OREPORT  H  11Ø   1P
02 O                                   68 'COMMISSION REPORT'
03 O        H  311   1P
04 O       OR        OV
05 O                                   35 'SALESMAN'
06 O                                   55 'DIST A'
07 O                                   65 'DIST B'
08 O                                   75 'DIST C'
09 O                                  11Ø 'TOTAL'
10 O        D  2     Ø1
11 O                         NAME       45
12 O                         COMMA 1B   55
13 O                         COMMB 1B   65
14 O                         COMMC 1B   75
15 O                         MANTOT1B  11Ø
16 O        T        LR
17 O                         TOTALA1    55
18 O                                   56 '*'
19 O                         TOTALB1    65
20 O                                   66 '*'
   O                         TOTALC1    75
   O                                   76 '*'
   O
```

Figure 5-60 (Part 4 of 4). Sales Commission Program (Using a Subroutine)

Using Valid Subroutine Operations

Any operation code that can be used in calculations can be used in a subroutine except BEGSR and ENDSR. This means that you can use all arithmetic, compare and testing, move look-up, EXSR, and branching operations.

There are limitations on some of the operations:

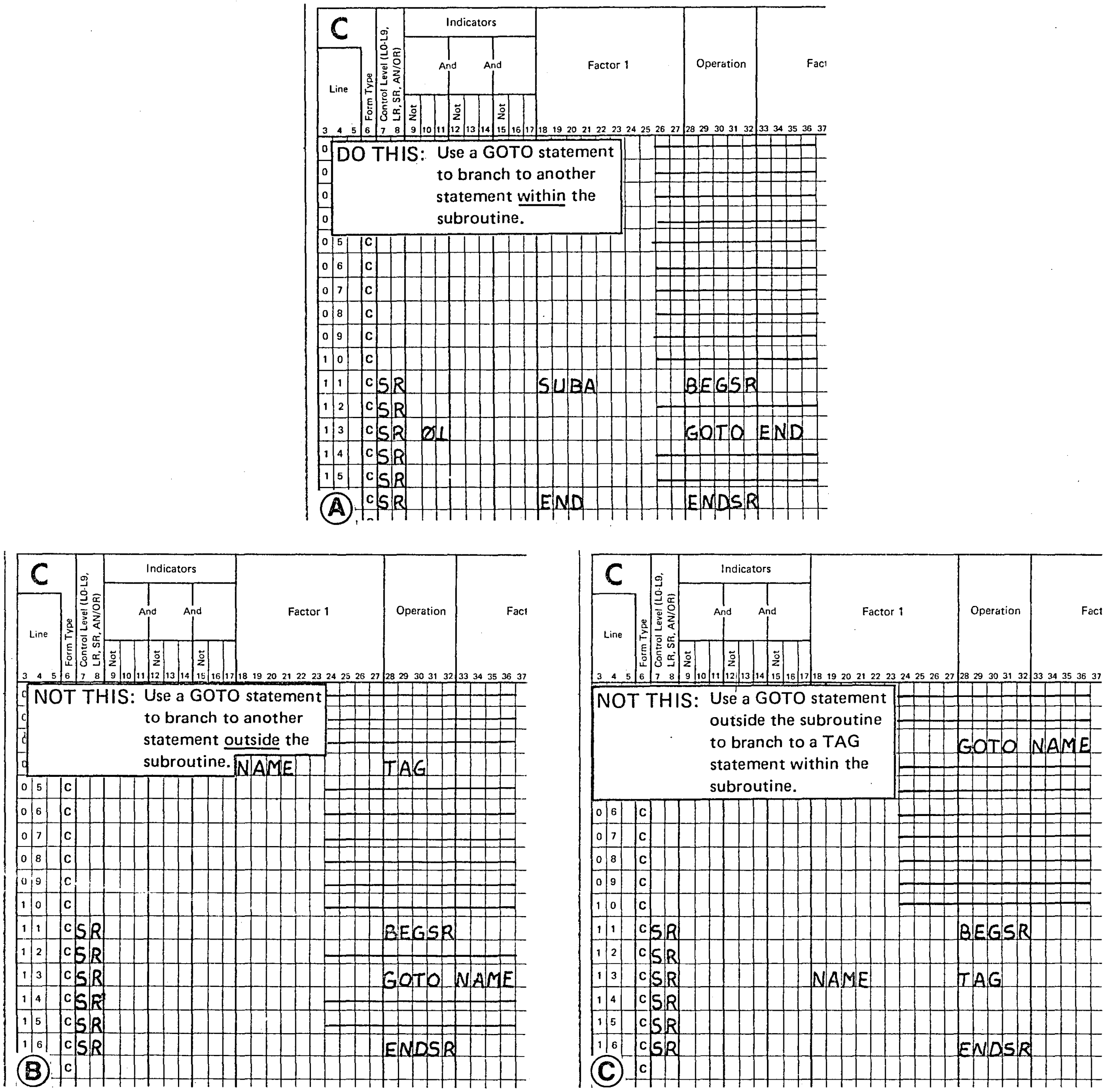
1. You may only branch to another statement *in the subroutine* when using the GOTO statement (Figure 5-61, insert A).

2. You may branch to the ENDSR statement if you put a name in Factor 1 of the ENDSR statement.

3. You may not branch to a statement outside of the subroutine (Figure 5-61, insert B).

4. You cannot branch to a TAG within the subroutine from a GOTO outside of the subroutine (Figure 5-61, insert C).

5. You cannot have a subroutine coded within another subroutine. However, one subroutine can call another subroutine. This means that within one subroutine you may have an EXSR statement (Figure 5-62). A subroutine, however, cannot call itself and cannot call the subroutine which called it.



A — DO THIS: Use a GOTO statement to branch to another statement within the subroutine.

```
11 C SR        SUBA      BEGSR
12 C SR
13 C SR Ø1               GOTO END
14 C SR
15 C SR
16 C SR        END       ENDSR
```

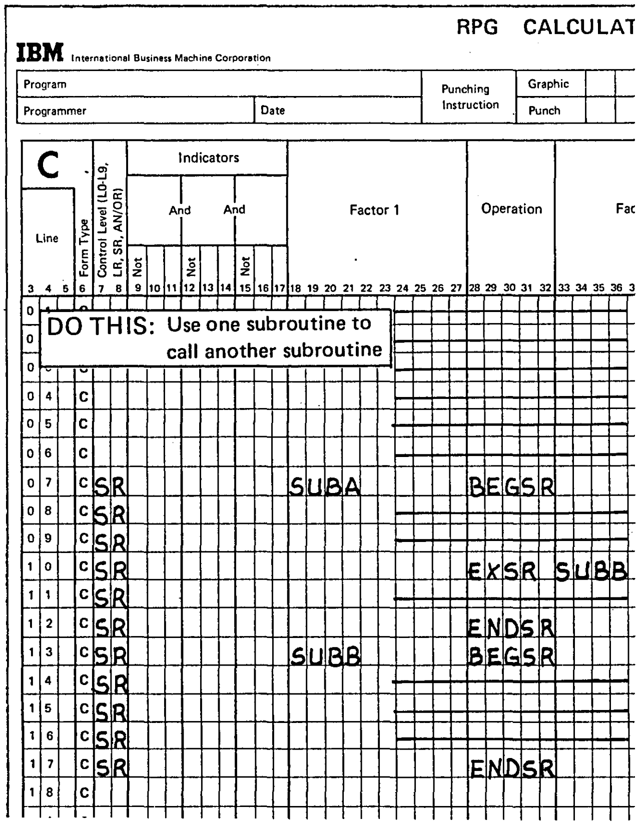
B — NOT THIS: Use a GOTO statement to branch to another statement outside the subroutine.

```
04 C           NAME      TAG
11 C SR                  BEGSR
12 C SR
13 C SR                  GOTO NAME
14 C SR
15 C SR
16 C SR                  ENDSR
```

C — NOT THIS: Use a GOTO statement outside the subroutine to branch to a TAG statement within the subroutine.

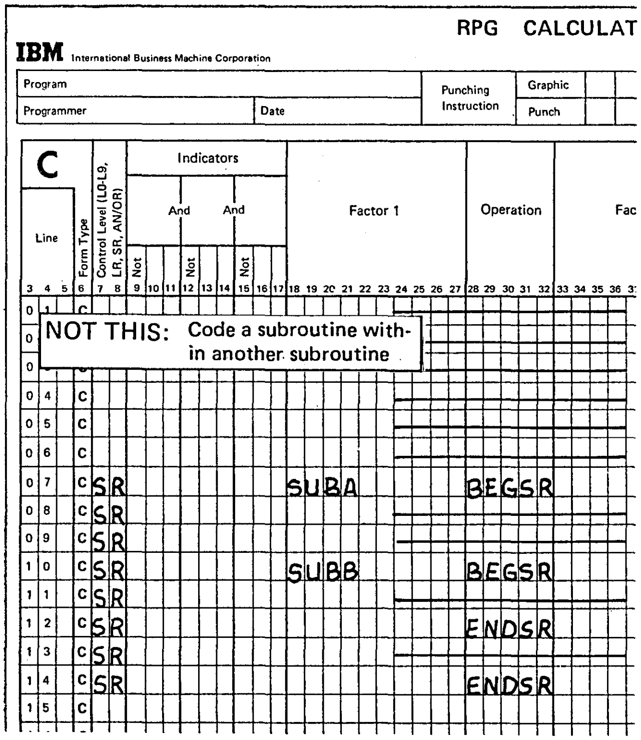
```
03 C                     GOTO NAME
11 C SR                  BEGSR
12 C SR
13 C SR        NAME      TAG
14 C SR
15 C SR
16 C SR                  ENDSR
```

Figure 5-61. Branching Within a Subroutine


RPG CALCULAT
IBM International Business Machine Corporation
Program
Programmer
Date
Punching Instruction
Graphic
Punch
C
Line
Form Type
Control Level (L0-L9, LR, SR, AN/OR)
Indicators
And
And
Not
Not
Not
Factor 1
Operation
Fac

DO THIS: Use one subroutine to call another subroutine

```
04 C
05 C
06 C
07 CSR         SUBA      BEGSR
08 CSR
09 CSR
10 CSR                   EXSR SUBB
11 CSR
12 CSR                   ENDSR
13 CSR         SUBB      BEGSR
14 CSR
15 CSR
16 CSR
17 CSR                   ENDSR
18 C
```


RPG CALCULAT
IBM International Business Machine Corporation
Program
Programmer
Date
Punching Instruction
Graphic
Punch
C
Line
Form Type
Control Level (L0-L9, LR, SR, AN/OR)
Indicators
And
And
Not
Not
Not
Factor 1
Operation
Fac

NOT THIS: Code a subroutine within another subroutine

```
04 C
05 C
06 C
07 CSR         SUBA      BEGSR
08 CSR
09 CSR
10 CSR         SUBB      BEGSR
11 CSR
12 CSR                   ENDSR
13 CSR
14 CSR                   ENDSR
15 C
```

Figure 5-62. Using EXSR Within a Subroutine

Conditioning Subroutine Statements

Any indicator which is valid in columns 9-17 can be used to condition an operation within the subroutine. That operation will then be performed only when the conditions established by the indicators are satisfied. The BEGSR and ENDSR operation code, however, cannot be conditioned by any indicators.

The EXSR statement can also be conditioned by indicators. In this case, the *entire* subroutine will be performed only when conditions for the EXSR statement are met. For example, in Figure 5-63, insert A, the subroutine will be performed only if MR is on.

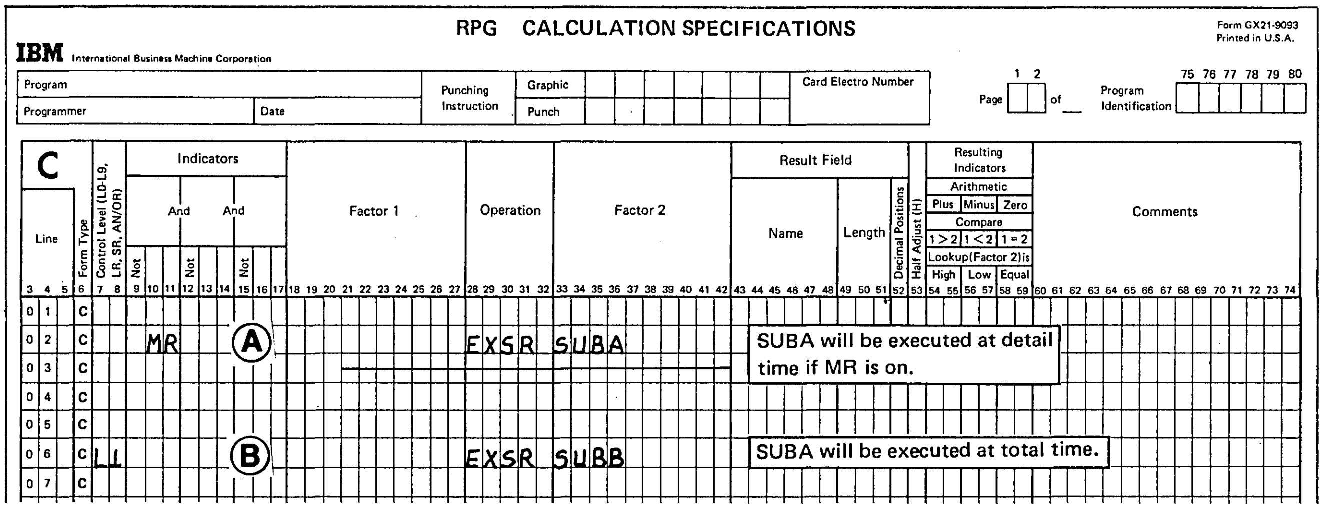
Control level indicators cannot be used to condition statements within a subroutine since SR must appear in columns 7-8. The indicators used on the EXSR statement determine whether the *entire* subroutine is performed at detail time or at total time (Figure 5-63, insert B).

RPG II LINKAGE EDITOR

The output of the RPG II compiler becomes input to the linkage editor, which builds the load (object) module for later execution. The module can be cataloged into the library, punched into cards, or written to diskette.

The linkage editor combines the translated RPG II source code with the system library modules necessary to complete the program. The map of the linkage editor's output is printed following the compiler diagnostic messages (if present). The map also indicates the amount of internal storage used by portions of a program.

The RPG II compiler used on Models 4, 6, 8, 10, or 12 includes its own linkage editor, called the RPG II Linkage Editor. On Model 15, the RPG II compiler uses the overlay linkage editor that is part of the SCP. The following refers to the RPG II Linkage Editor, although conceptually, it is identical to the overlay linkage editor.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification
Programmer | Date

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field	Comments
01	C							
02	C		MR (A)		EXSR	SUBA		
03	C							
04	C							
05	C							
06	C	L1	(B)		EXSR	SUBB		
07	C							

SUBA will be executed at detail time if MR is on.

SUBA will be executed at total time.

Figure 5-63. Conditioning Calculations Within a Subroutine

Linkage Editor Map

Figure 5-63.1 illustrates the information available on the RPG II Linkage Editor map.

The *Start Addr* column gives the physical location in storage of each module. Figure 5-63.1 shows the modules in ascending sequence by address (which may not always be the case). The root section indicates the beginning of the program just after the system supervisor (0E00, in this case). The root section is 1,000 hexadecimal bytes long (see *Code Length* column); therefore, the next available location is 1E00. Note that the next higher address in the example is 1E00 for the input mainline section.

Start Addr	Name if Overlay	Code Length	Name	Title
0E00		1000	RGROOT	Root
1E00		0100	RGMAIN	Input Mainline
1F00		0100	RGSUBS	Record ID
2000		0050	RGSUBS	Input ctrl rtn
2050		0010	RGSUBS	Subseg
2060		0020	$$CSIP	5444 consec input
2080		0080	RGMAIN	Input fields
2100		0050	RGMAIN	Detail calcs
2150		0100	$$PGLC	Lokup routine
2250		0200	RGMAIN	Detail output
2450		0B00	RGSUBS	Constants

Figure 5-63.1. Linkage Editor Map

The *Code Length* column gives the storage requirements in hexadecimal. The *Name* column gives the name of system (and user) modules and designates the compiler-created segments (which begin with RG). The functions of modules are described, where practical, in the title column.

The root section contains I/O buffers, constants, and variables required to run RPG II programs. Subseg is a section that has functions that are not easily describable by the compiler. For example, a calculation subroutine is called a subseg.

The RGMAIN sections are the basic RPG II functions, such as input, detail calculations, total output, and so on. RGSUBS designates functions within the RGMAIN functions. The modules are listed in functional sequence; the subsections follow the main function. In Figure 5-63.1, the lokup routine follows the detail calcs section because a table look-up was performed in detail calculations. Similarly, the constants section, because it follows the detail output section, was defined on the output specifications for detail (or heading) lines.

Simple Overlays

An overlay structure begins if the load (object) module built by the linkage editor is larger than the available main storage in a machine (the size is specified on the H-card or is assumed to be that of the machine used for compilation). The linkage editor begins with low-usage modules such as execution time table dump and load routines, file open and close, LR calculations and output, and so on. It allocates an overlay fetch area large enough to hold the largest of the selected routines (or groups of routines) and calls for an overlay fetch routine, which will load into that overlay area the selected routines as they are needed. The overlay fetch routine checks to see if a requested overlay is already in the overlay area before going to the disk library to load it. This action saves time by avoiding unnecessary loads from disk.

With overlays, the root section occupies the lowest position in storage as before. It is followed by the overlay fetch routine. The overlay fetch area comes next, followed by the secondary root routine, or root 2. Root 2 is made up of those routines that do not have to be overlaid.

All the overlaid routines are given names in the *Name If Overlay* column. The names are $##nnn where nnn is a three-digit overlay number.

Figure 5-63.2 shows a storage map of a program and Figure 5-63.3 shows a library map of the same program. Figure 5-63.4 is a diagram that illustrates the maps shown in Figures 5-63.1 and 5-63.2. The library map indicates the overlay size in the *# Text Sectors* column. You can see in Figure 5-63.3 that overlay #2 ($##002) is the largest, occupying seven sectors; this is equivalent to 0700 bytes in hexadecimal (all storage requirements will be stated in four hexadecimal digits). The storage map shows the overlay fetch area of 0700 bytes. It also indicates that overlay #2 contains the total calculations and is actually only 0630 bytes long (0600 for total calcs, 0020 for constants, and 0010 for exception output). Therefore, overlay #2 occupies seven sectors: six are full and the seventh has only 0030 bytes of code in it. If the user wants to reduce the size of the program's overlay area, he would first try to reduce the size of total calculations.

Start Addr	Name if Overlay	Code Length	Name	Title
0E00		1000	RGROOT	Root
1E00		0100	RGSUBS	Overlay fetch routine
1F00		0700	RGSUBS	Overlay fetch area
2600		0100	RGMAIN	Input mainline
2700		0100	RGMAIN	Input fields
1F00	$##001	0200	RGMAIN	Detail calcs
2100	$##001	00A0	RGSUBS	Constants
1F00	$##002	0600	RGMAIN	Total calcs
2500	$##002	0020	RGSUBS	Constants
2520	$##002	0010	RGSUBS	Exception

Figure 5-63.2. Storage Map

Overlay Name	Relative Start C/T/S			#Text Sectors	Start Address
$##001	00	00	16	03	1F00
$##002	00	01	02	07	1F00
$##003	00	01	0A	06	1F00
$##004	00	01	11	04	1F00

Figure 5-63.3. Library Map

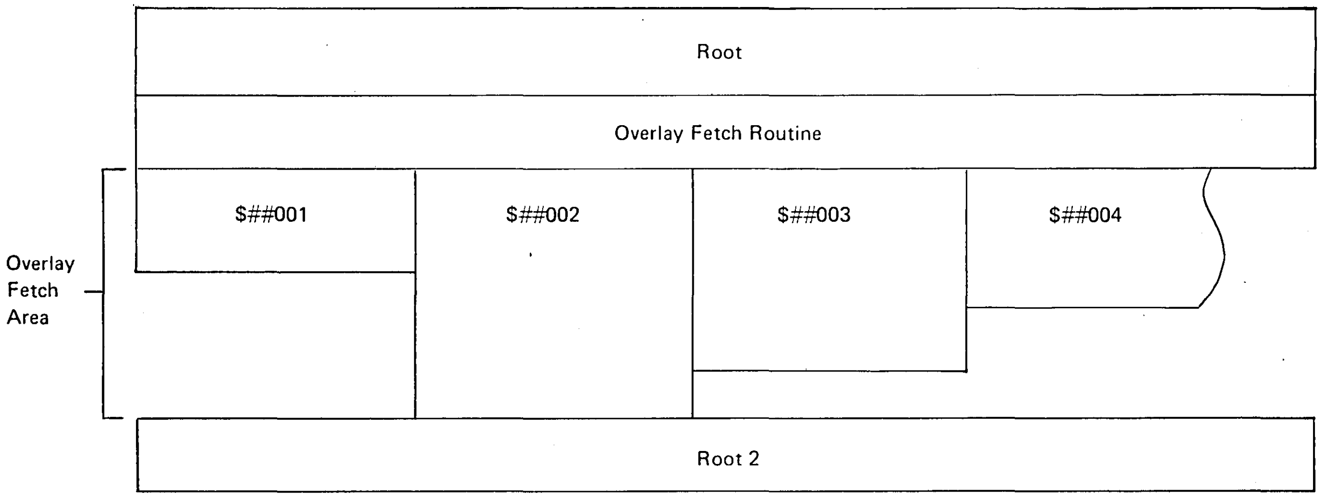


Figure 5-63.4. Diagram of Linkage Editor Map and Storage Map

The *Relative Start C/T/S* column gives relative library locations in cylinder, track, and sector for the overlays. This information does not pertain to this discussion.

Special Open

Sometimes a storage map will not list an overlay #1 but will have other overlays starting with $##002. This indicates that the special open has been implemented. The user cannot directly cause this to happen. The linkage editor provides this feature when it reduces a program's storage requirements. Special open does not affect throughput.

When the open/close overlay is the largest one, special open is implemented, rather than having that code increase the storage required for the rest of the program. Special open causes some of the unoverlaid code in root 2 to overlay the open routine after the files have been opened; it overlays some of root 2 with close at end-of-job.

Overlay Starting Addresses

As the maps in Figure 5-63.5 show, there are two starting addresses for overlays. The main starting address for the illustration is 1F00. The starting address for the suboverlay is always higher; in this case, it is 2300.

Storage Map

Start Addr	Name if Overlay	Code Length	Name	Title
1F00	$##002	0200	RGMAIN	Detail calcs
2100	$##002	0100	RGSUBS	Transfer vector
2200	$##002	0002	RGSUBS	Contants
2300	$##003	0100	$$PGRI	Reset Resulting indr
2400	$##003	0100	RGSUBS	Subseg
2300	$##004	0300	RGSUBS	Exception

Library Map

Overlay Name	Relative Start C/T/S	#Text Sectors	Start Address
$##001		07	1F00
$##002		04	1F00
$##003		02	2300
$##004		03	2300
$##005		03	1F00
$##006		04	2300
$##007		02	2300

Figure 5-63.5. Storage and Library Maps

$##002 is a main overlay with suboverlays. There are two indications of this: first, it includes a module RGSUBS (which is titled transfer vector), and second, it is followed on the maps by overlays with higher starting addresses.

The size of the overlay fetch area is the larger of (1) the largest overlay with suboverlays plus the largest suboverlay, or (2) the largest overlay without suboverlays. In Figure 5-63.5, the size of the overlay fetch area is equal to the sum of the sizes of $##002 and $##006. This is not obvious from the maps, but the block diagram (Figure 5-63.6) drawn from the maps makes this clear. Overlay 2 is the largest main overlay with suboverlays, so it determines the starting address for all suboverlays. Overlay 6 is the largest suboverlay. Together, overlays 2 and 6 require 8 sectors or 0800 bytes, which is greater than the 7 sectors (or 0700 bytes) for overlay 1.

If a user wants to reduce the size of the overlay fetch area, he could reduce the size of either $##002 or $##006. One sector, however, is all the user can expect to save because $##001 is just one sector smaller than the overlay fetch area.

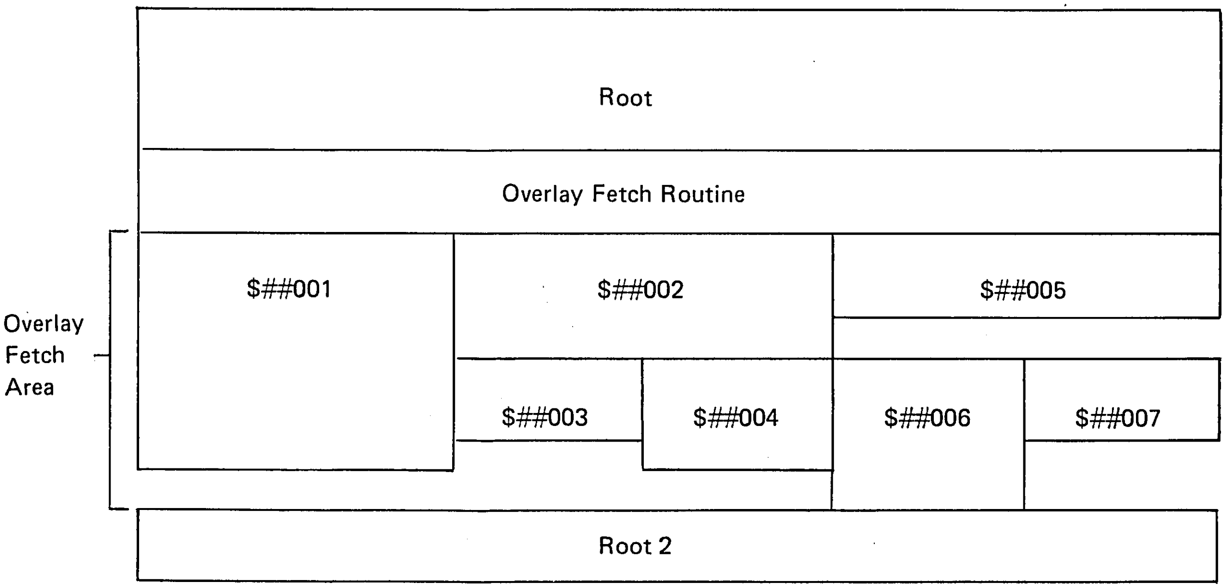


Figure 5-63.6. Diagram of Storage and Library Maps

Fitting Available Storage

The linkage editor removes modules from root 2, placing them into the overlay structure until the program fits into available storage (or until there are no more modules to overlay, which means the program is too large to fit). After a fit is obtained, there may be unused space in storage. If there is, an attempt is made to find small modules that can be removed from the overlay structure and placed back into that free space.

This process of moving overlaid modules back into main storage might give you a *missing overlay*, that is, a suboverlay without a main overlay. In the storage map, you can sometimes notice unoverlaid modules among the overlays. This is a result of moving modules from main storage to an overlay and back again.

The linkage editor moves code rather than allowing available storage go unused. Having code moved back to main storage can improve speed of execution, but is not under the control of the user. The linkage editor moves the code when it can.

RPG II Logic and Function Shifting

The basic functional areas of RPG II logic are summarized in the following sequence:

1. Detail Output—Detail and heading output (except for overflow lines) are performed.

2. Physical Input—Records are read from primary and secondary files, as necessary, and the ID indicator is set.

3. Total Calculations—All calculations with L and a number in columns 7 and 8 are performed. This includes all exception output specifications if EXCPT is performed, or all required input specifications for READs or CHAINs performed, or any user subroutines called with EXSR.

4. Total Output—Total output (except for overflow lines) is performed. You should note that total calculations and output are attempted on every cycle.

5. Overflow Output—All output lines with overflow indicators conditioning them are performed.

6. Logical Input—Data read in Step 2 is moved to the fields named on the input sheet, and the corresponding field indicators are set.

7. Detail Calculations—All calculations without L and a number in columns 7 and 8 are performed. See Step 3 for more information.

These functional areas can be identified by the titles of routines in the storage map. A user can reduce the size of any of these areas by one of three methods:

- Removing function from the program
- Using more efficient coding
- Shifting function from one general area to another

The removal of function from a program often results in a second program being written to perform those functions. Some system designs include luxuries that are not essential to the job. These functions may be removed from the program.

Functions that cannot be removed can often be moved to another area. Some total calculations can be moved easily to detail calculations.

The following multiply statement could be moved to detail calculations by conditioning it with the 01 indicator, just like the add, and deleting the L1 indicator on the MULT statement. The results will be identical. Similarly, some calculations can be moved from detail to total; however, a new record ID indicator will be on so the user may have to use an extra indicator.

```
C      01      AMT      ADD      TOTAL      TOTAL
CL1            TOTAL    MULT     RATE       DISCNT
```

Primary and secondary input are done at input time. READ and CHAIN are performed during calculations. If calculations are too big, a file processed with the CHAIN operation code might be first sorted and then matched as a secondary file against the primary. If input is too big, a secondary file could be made a demand file and processed with the READ operation code during calculations.

Output, like calculations, can be shifted between detail and total calculations. Even overflow output can be moved to the following detail or total calculations by using a numeric indicator, which is SETON conditioned by the overflow indicator. EXCPT is output performed during calculations. Some output functions may be moved from output to calculations (either detail or total).

Any calculation overlay or suboverlay that does an EXCPT will contain all E output specifications. If a suboverlay has EXCPT in it and is too large, that operation can usually be moved to another subroutine. If the subroutines are large, they can usually be divided into several smaller subroutines.

Calculation sections cannot be divided by the linkage editor to be separate overlays; subroutines can. They will become suboverlays to the main detail or total calculations overlay. If one subroutine calls another subroutine, both will appear in the same suboverlay. To divide them up, the first subroutine must return to regular calculations from which the second subroutine can be called.

Using shared I/O for disk files can save a great deal of space in some programs. With the resulting smaller overlay structure, performance may be improved. Alternatively, larger disk file buffers and double buffering could be specified. These two techniques use extra storage (resulting in more overlays) but may still perform faster because of the speed improvement in disk operations they provide. Selection between larger disk I/O buffers or shared buffers must be made by trial for each program.

Information about the overlay linkage editor can be found in the *IBM System/3 Overlay Linkage Editor Reference Manual*, GC21-7561.

SPECIAL USES OF CONTROL LEVEL INDICATORS

At this time, you should be familiar with the normal use of control level indicators, L1-L9. You know that by assigning them on the Input sheet and using them on the Calculation and Output Specifications sheets you can do total operations.

In this section, you will learn to work with a special internal control level indicator L0. You will also learn to use L1-L9 indicators to condition calculations and to perform group printing.

Internal Control Level Indicator L0

L0 is a unique control level indicator which is always on. You cannot assign it to a field, as you do L1-L9, by entering it in columns 59-60 of the Input sheet. But, you can use it to condition a calculation or output operation. The operation so conditioned will be done at total time for every program cycle, since L0 is always on.

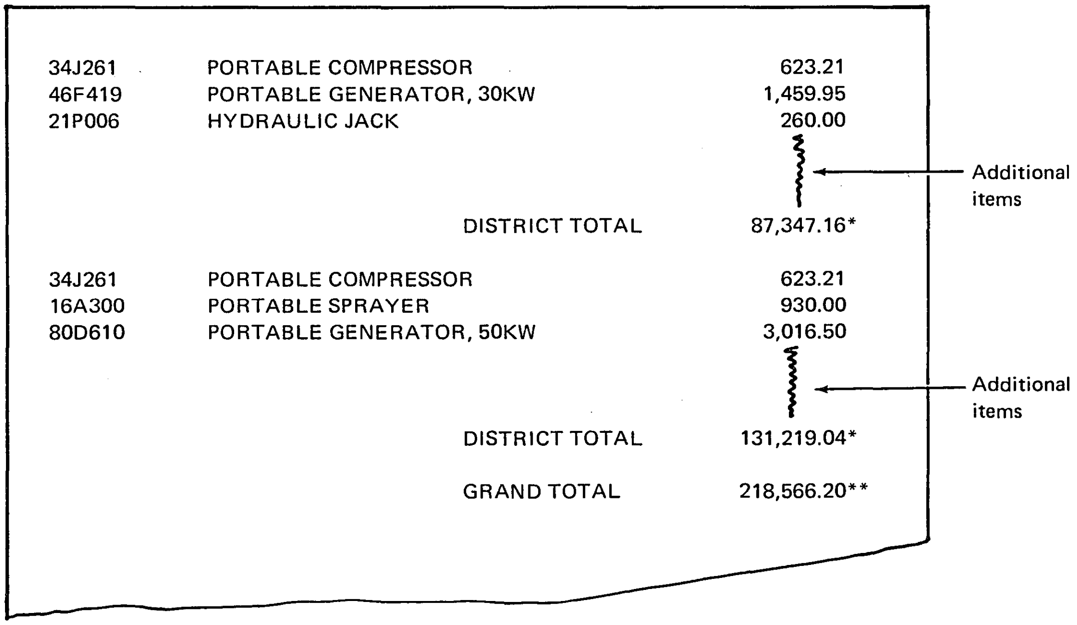
The main purpose of the L0 indicator is to allow you to specify total operations when indicators L1-L9 are not available or when they cannot accomplish the job.

Suppose you want to print the report shown in Figure 5-64. The report is a listing of items a company has sold in each of its two districts. The report includes total sales for each district (DISTOT) and a grand total of sales in both districts (GDTOT). The input records are grouped by district with District 1 records preceding District 2. Each record contains an item number field (ITEM), an item description (DESCR), and an item cost field (COST), as shown in Figure 5-65.

A record identification code in position 1 indicates in which district the item was sold. Normally, the field containing the record identification code could be used as a control field in the printing of district totals. However, each district has more than one record identification code. District 1 uses either a 1 or an M as an identifier and District 2 uses either a 2 or an N. Since the contents of the record identification field can change without actually having a change in district, this field cannot be used as a control field. Neither, of course, can the ITEM, DESCR, or COST fields be used as control fields.

Artificial Control Breaks

When it is necessary to do total operations but no control fields are available to cause a control break, you can use L0 to cause an artificial control break. Figure 5-66, insert B, shows the use of L0 to permit total calculation operations.



```
34J261      PORTABLE COMPRESSOR                               623.21
46F419      PORTABLE GENERATOR, 30KW                        1,459.95
21P006      HYDRAULIC JACK                                    260.00

                                   DISTRICT TOTAL          87,347.16*

34J261      PORTABLE COMPRESSOR                               623.21
16A300      PORTABLE SPRAYER                                  930.00
80D610      PORTABLE GENERATOR, 50KW                        3,016.50

                                   DISTRICT TOTAL         131,219.04*

                                   GRAND TOTAL            218,566.20**
```

Additional items
Additional items

Figure 5-64. Format of District Sales Listing

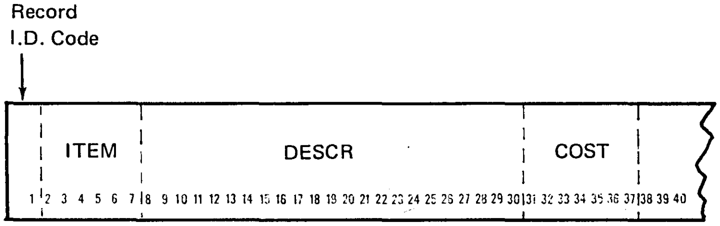
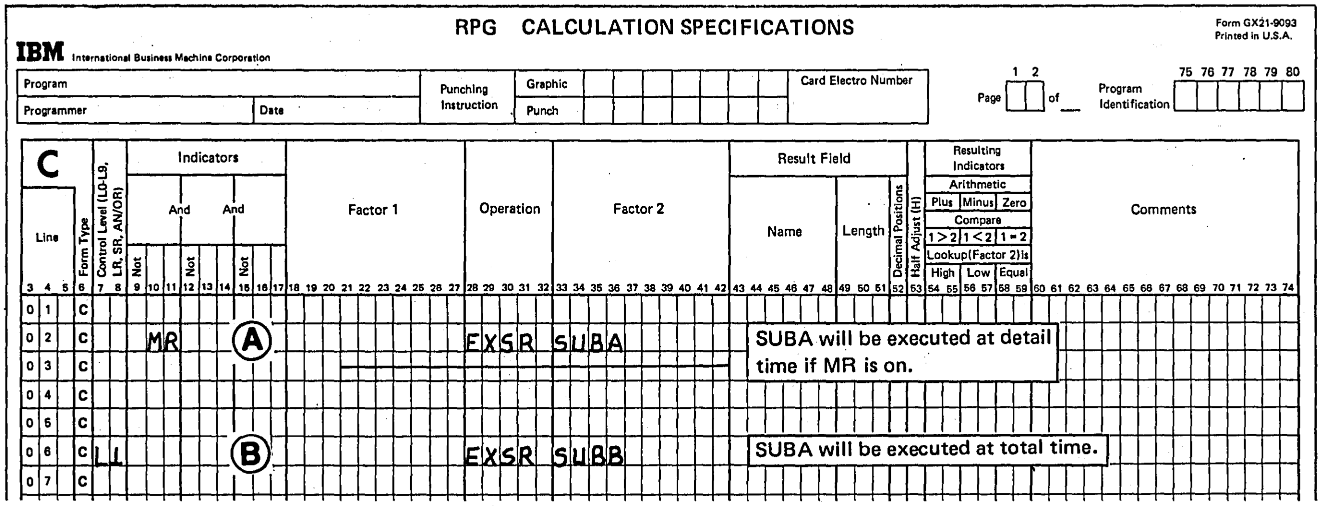


Figure 5-65. Format of Sales Record

This page is intentionally left blank.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

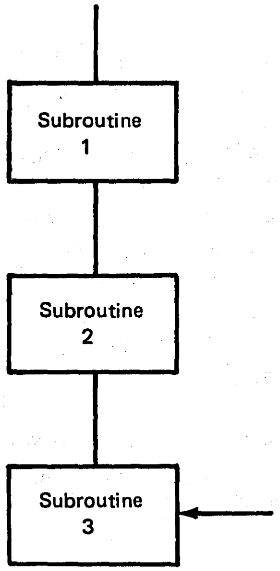
Form GX21-9093
Printed in U.S.A.

Program		Punching Instruction	Graphic	Card Electro Number	Page 1 2 of
Programmer	Date		Punch		Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field	Comments
01	C							
02	C		MR (A)		EXSR	SUBA	SUBA will be executed at detail time if MR is on.	
03	C							
04	C							
05	C							
06	C	L1	(B)		EXSR	SUBB	SUBA will be executed at total time.	
07	C							

Figure 5-63. Conditioning Calculations Within a Subroutine

You must give priority to subroutines to determine which subroutines, rather than object cycle routines, should be stored on disk. Those subroutines used infrequently should be the first routines stored in the object library. Priority is established through the order in which the subroutines appear at compilation time. The last subroutine in your source program will be the first subroutine stored in the object library. Consequently, you should place an infrequently used subroutine as the last subroutine in your source program:



The last subroutine in your source program is the first subroutine stored on disk.

SPECIAL USES OF CONTROL LEVEL INDICATORS

At this time, you should be familiar with the normal use of control level indicators, L1-L9. You know that by assigning them on the Input sheet and using them on the Calculation and Output-Format sheets you can do total operations.

In this section, you will learn to work with a special internal control level indicator L0. You will also learn to use L1-L9 indicators to condition calculations and to perform group printing.

Internal Control Level Indicator L0

L0 is a unique control level indicator which is always on. You cannot assign it to a field, as you do L1-L9, by entering it in columns 59-60 of the Input sheet. But, you can use it to condition a calculation or output operation. The operation so conditioned will be done at total time for every program cycle, since L0 is always on.

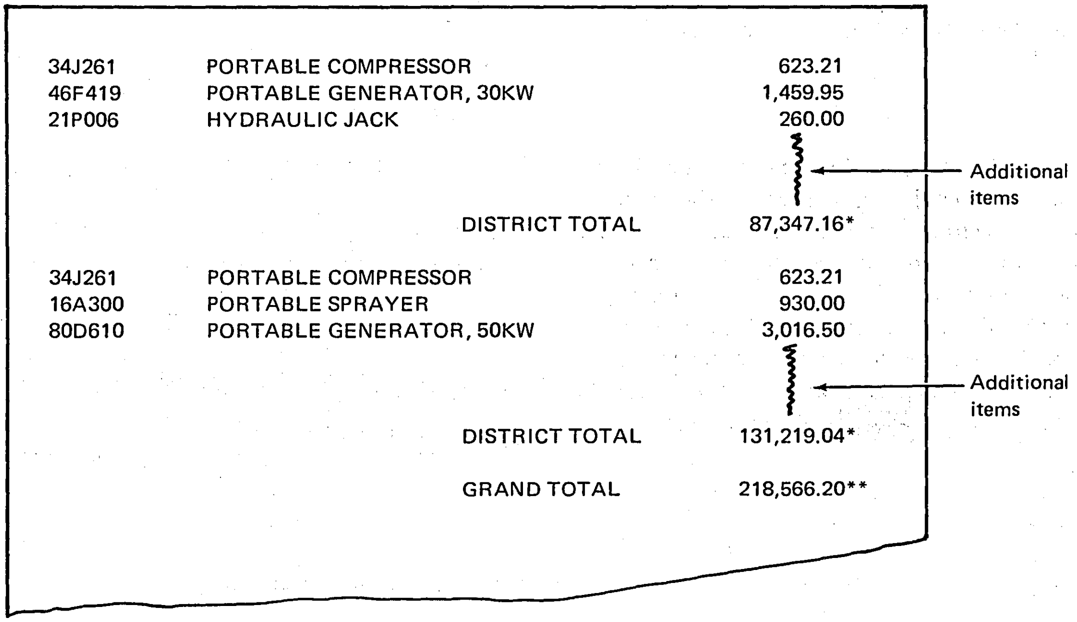
The main purpose of the L0 indicator is to allow you to specify total operations when indicators L1-L9 are not available or when they cannot accomplish the job.

Suppose you want to print the report shown in Figure 5-64. The report is a listing of items a company has sold in each of its two districts. The report includes total sales for each district (DISTOT) and a grand total of sales in both districts (GDTOT). The input records are grouped by district with District 1 records preceding District 2. Each record contains an item number field (ITEM), an item description (DESCR), and an item cost field (COST), as shown in Figure 5-65.

A record identification code in position 1 indicates in which district the item was sold. Normally, the field containing the record identification code could be used as a control field in the printing of district totals. However, each district has more than one record identification code. District 1 uses either a 1 or an M as an identifier and District 2 uses either a 2 or an N. Since the contents of the record identification field can change without actually having a change in district, this field cannot be used as a control field. Neither, of course, can the ITEM, DESCR, or COST fields be used as control fields.

Artificial Control Breaks

When it is necessary to do total operations but no control fields are available to cause a control break, you can use L0 to cause an artificial control break. Figure 5-66, insert B, shows the use of L0 to permit total calculation operations.



```
34J261     PORTABLE COMPRESSOR                          623.21
46F419     PORTABLE GENERATOR, 30KW                   1,459.95
21P006     HYDRAULIC JACK                               260.00

                                 DISTRICT TOTAL      87,347.16*

34J261     PORTABLE COMPRESSOR                          623.21
16A300     PORTABLE SPRAYER                             930.00
80D610     PORTABLE GENERATOR, 50KW                   3,016.50

                                 DISTRICT TOTAL     131,219.04*

                                 GRAND TOTAL        218,566.20**
```

Additional items
Additional items

Figure 5-64. Format of District Sales Listing

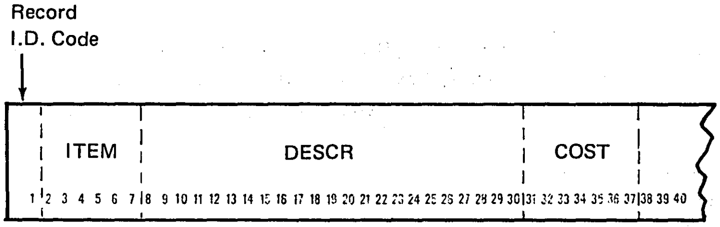
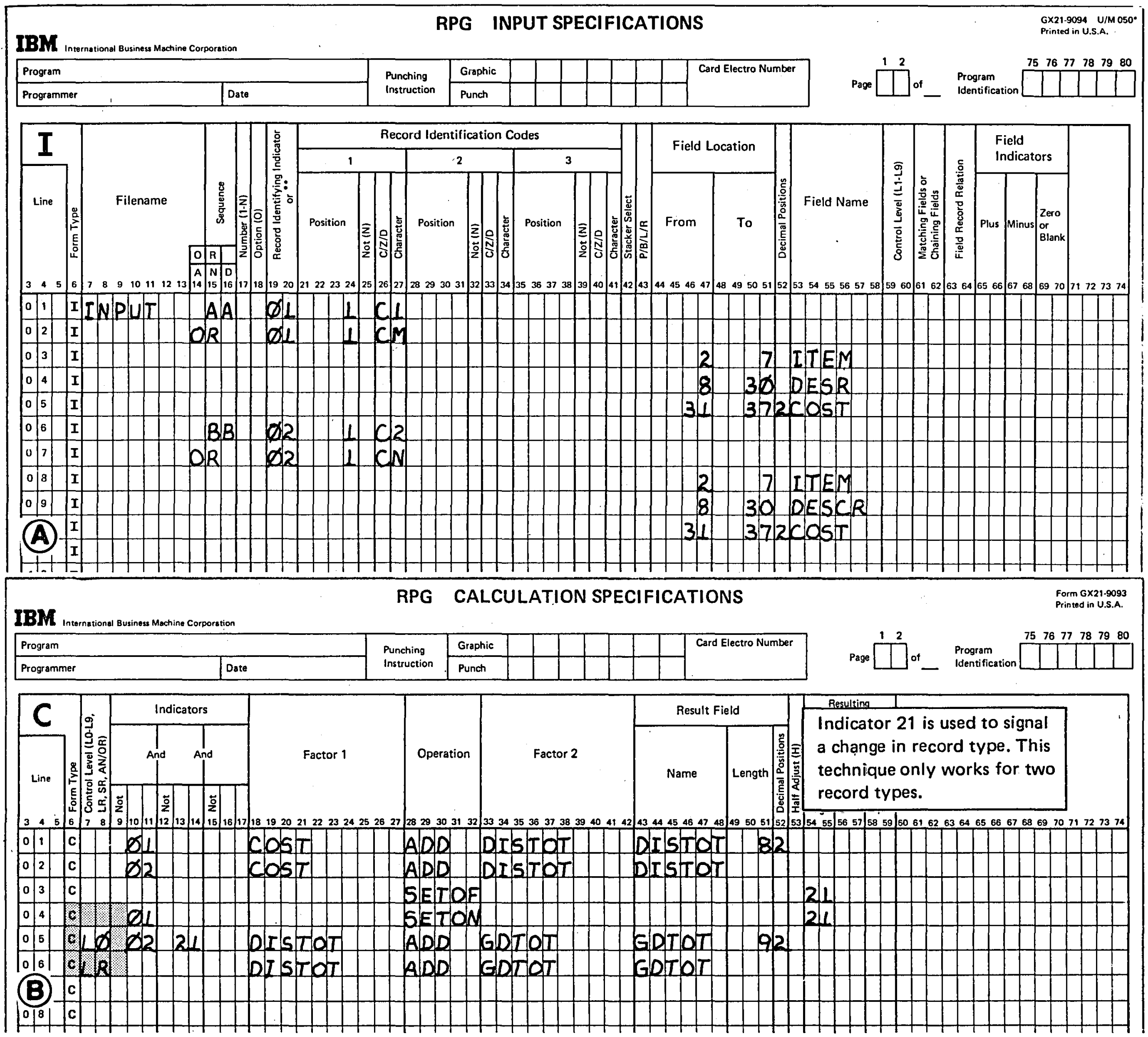
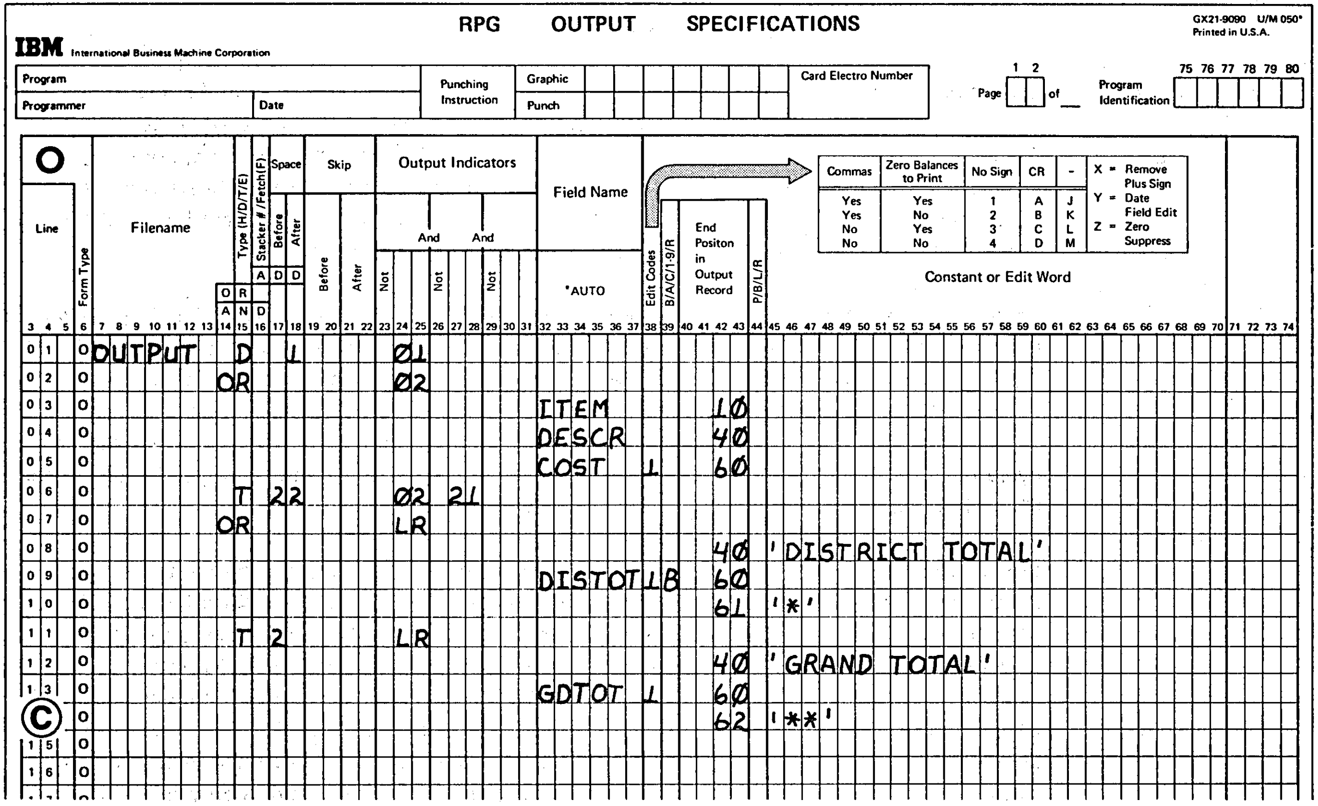


Figure 5-65. Format of Sales Record





RPG OUTPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Programmer | Date

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

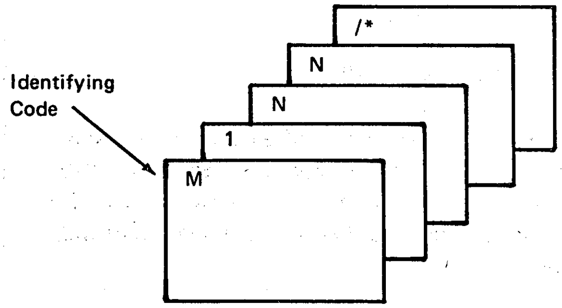
X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line | Form Type | Filename | Type (H/D/T/E) | Stacker #/Fetch(F) | Space Before After | Skip Before After | Output Indicators And And Not | Field Name *AUTO | Edit Codes | B/A/C/1-9/R | End Positon in Output Record | P/B/L/R | Constant or Edit Word

```
01 OOUTPUT  D  1     Ø1
02 O      OR         Ø2
03 O                         ITEM      1Ø
04 O                         DESCR     4Ø
05 O                         COST  1   6Ø
06 O        T 22     Ø2 21
07 O      OR         LR
08 O                                   4Ø 'DISTRICT TOTAL'
09 O                         DISTOT1B  6Ø
10 O                                   61 '*'
11 O        T 2      LR
12 O                                   4Ø 'GRAND TOTAL'
13 O                         GDTOT 1   6Ø
   O                                   62 '**'
15 O
16 O
```

Figure 5-66 (Part 2 of 2). Calculations Using an Artificial Control Break (L0 Indicator)

Assume that the following five records are read:



The operations performed on these five records are as follows:

Record	*Indicators On*	*Operations Performed*
(1)	L0	01 turns on. No total operations are performed because conditions in lines 5-6 (Calculations sheet) are not met. (Remember that operations conditioned by control level indicators in columns 7-8 are performed first, but are bypassed on the first RPG II cycle.) COST is added to DISTOT. 21 is set on. ITEM, DESCR, and COST are printed out. 01 is turned off. 21 remains on.
(2)	L0, 21	01 is turned on. No total operations are performed. COST is added to DISTOT. ITEM, DESCR, and COST are printed out. 01 is turned off. 21 remains on.
(3)	L0, 21	02 turns on. DISTOT is added to GDTOT. (Conditions for the total operation in line 5 have been met.) DISTOT is printed out. COST is added to DISTOT. 21 is set off. ITEM, DESCR, and COST are printed out. 02 is turned off.
(4)	L0	02 is turned on. No total operations are performed. COST added to DISTOT. ITEM, DESCR, and COST are printed out. 02 is turned off.
(5)	LR	DISTOT added to GDTOT (LR indicator is on). DISTOT and GDTOT printed out.

Using Control Level Indicators as Calculation Conditioning Indicators

Control level indicators are normally entered in columns 7-8 of the Calculation sheet where they specify which calculations are to be done at total time. They may, however, be used in columns 9-17 also where they indicate which detail operations are to be done on the first record of a control group.

Control level indicators are turned on near the beginning of the program cycle if the contents of the control field on the record just read are different from the contents of the previous control field. Since the indicator is not turned off until the end of the cycle, it is on during total and detail time. Thus, it is available to use as a conditioning indicator during detail time as well as total time.

When an operation is not conditioned by control level indicators specified in columns 7-8 of the Calculation sheet, the operation is done at detail time. If the operation is conditioned by a control level indicator specified in columns 9-17, and not in 7-8 the operation is still done at detail time, but only when the control level indicator is on. And when is the control level indicator on? Only during the processing of the first record in a control group as only the first record in a new group causes a control break.

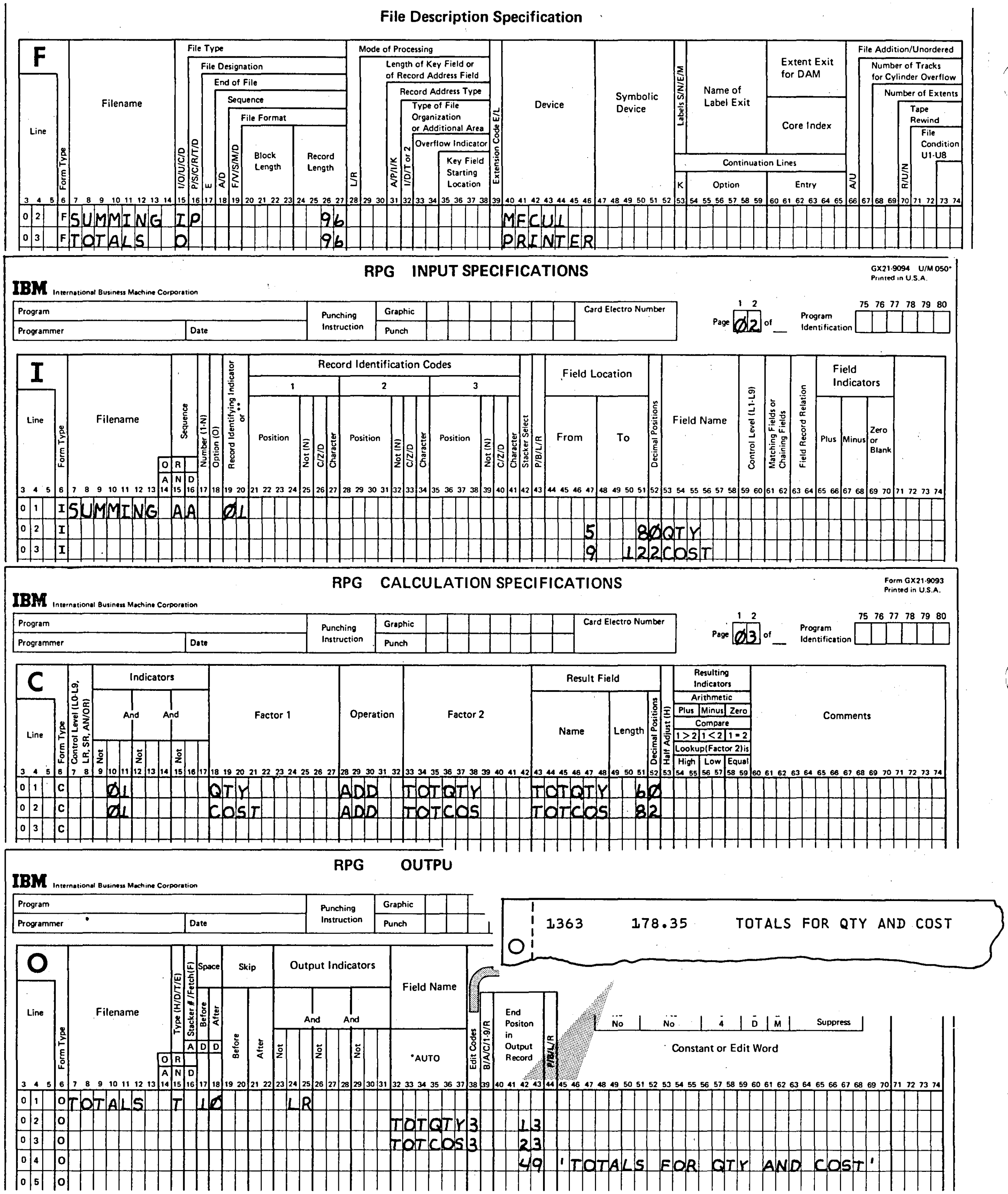
Group Printing

In group printing, data from groups of input records is summarized and printed as totals on a report. Sometimes, a field on each record in an input file is accumulated and only the final total is printed. At other times, subtotals are created by adding the contents of a field from a certain group of records and printing the result.

Printing Only the Final Totals

It is possible to add the contents of a field from all data records, and print only the total accumulated. You may want to do this when you are not interested in any of the detail information, but you do need a summation of that information.

Figure 5-67 shows the coding for a program that finds the totals for two fields of information and prints only those totals. Notice, when printing only totals, the Input sheet need only contain those fields that will be used in the calculations or in the printing. The Output-Format sheet shows only a total line (T in column 15). That line is not printed until the last record has been processed (LR in columns 24-25). Then, one line is printed, containing the total quantity, total cost, and a constant (see insert on Output-Format sheet).



File Description Specification

```
02 FSUMMING IP         96            MFCU1
03 FTOTALS  O          96            PRINTER
```

RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation — GX21-9094 U/M 050* Printed in U.S.A. — Page Ø2

```
01 ISUMMING AA  Ø1
02 I                                        5   8ØQTY
03 I                                        9  122COST
```

RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation — Form GX21-9093 Printed in U.S.A. — Page Ø3

```
01 C   Ø1      QTY       ADD  TOTQTY    TOTQTY  6Ø
02 C   Ø1      COST      ADD  TOTCOS    TOTCOS  82
03 C
```

RPG OUTPU

IBM International Business Machine Corporation

```
01 OTOTALS  T 1Ø     LR
02 O                            TOTQTY3  13
03 O                            TOTCOS3  23
04 O                                     49 'TOTALS FOR QTY AND COST'
05 O
```

```
1363      178.35         TOTALS FOR QTY AND COST
```

Figure 5-67. Group Printing – Printing Only the Final Totals

Group Printing of Subtotals

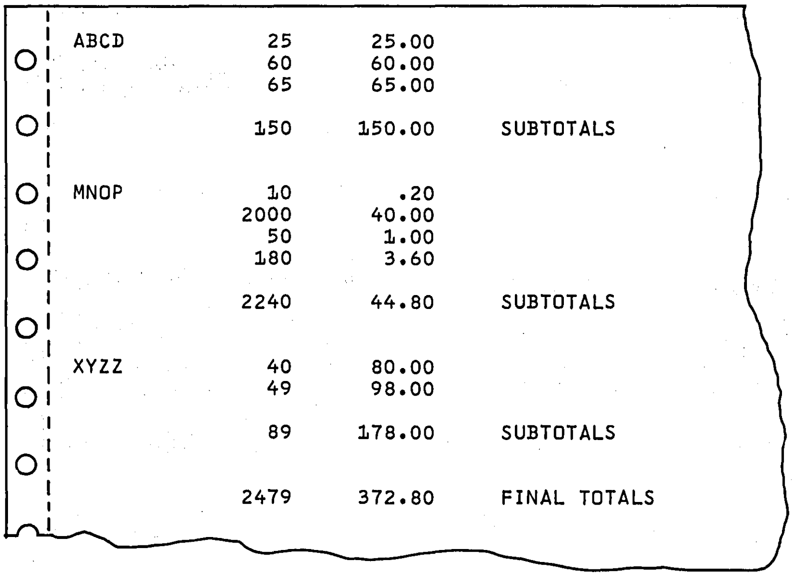
Figure 5-68 shows group printing of subtotals. The input records for the program contain item, quantity, and cost fields; they were previously sorted by like items.

Figure 5-69 shows the coding to produce the report. Since a subtotal is to be calculated for each different type of item, the field ITEM is designated as a control field on Input specifications. The calculations show that, as each input record is read, the values in the QTY and COST are accumulated into subtotal fields. At each control break, the subtotal fields, SUBQTY and SUBCOS, are accumulated into final total fields.

The Output-Format sheet in Figure 5-69 shows coding for three different types of printed lines:

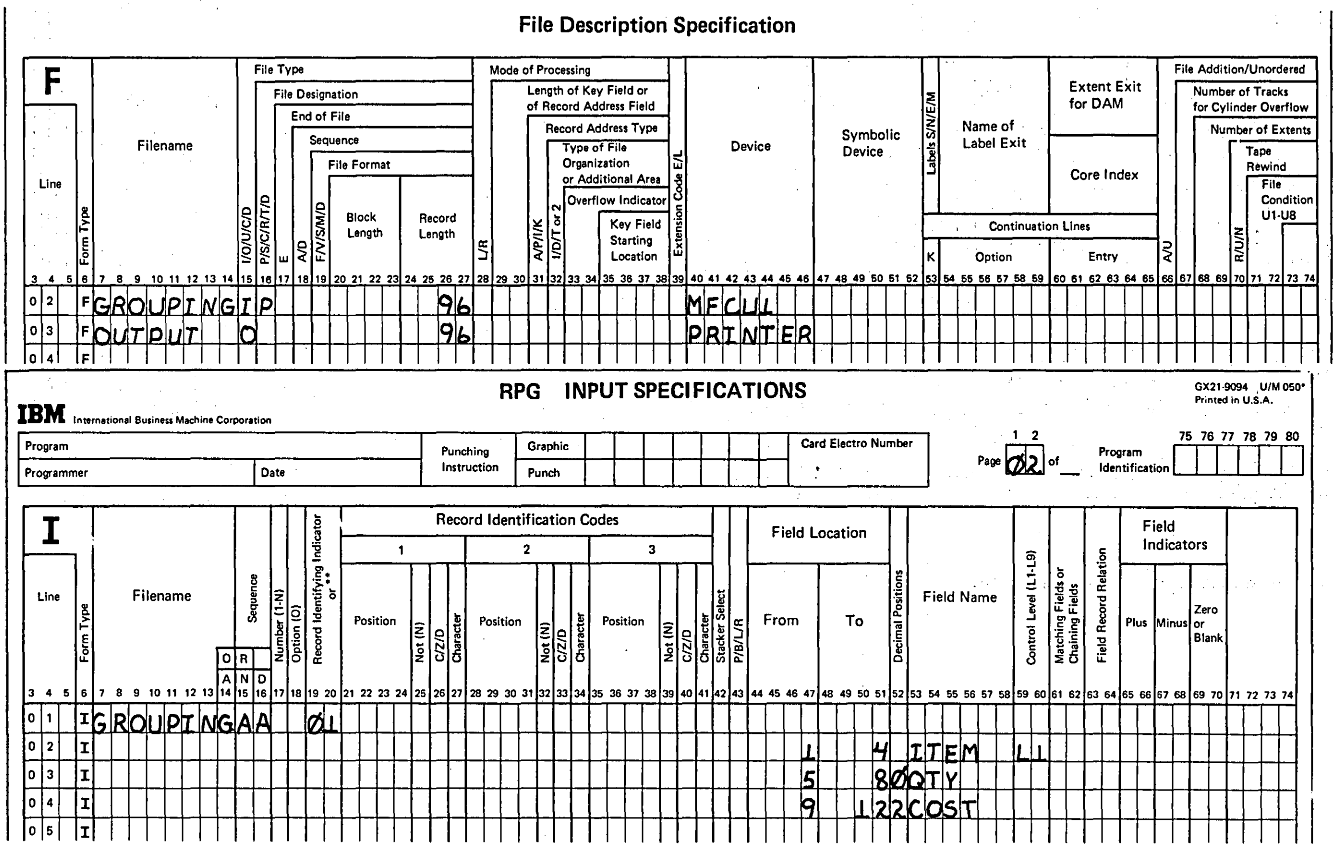
- A detail line, which is printed for each input record read. Note the use of a control level indicator on the ITEM field. This causes the field to be printed only for the first record of each control group (see Figure 5-68), making the report less cluttered and easier to read. This is known as *group indication* (see index entry).
- A total line, conditioned by L1, which is printed only after each complete group of like items was read, that is, after each control break.
- A final total line, conditioned by LR, which is printed only once, after all records were processed.

If you want the subtotals to be the sums of only the quantities and cost in one section (for instance, only those quantities and costs under item ABCD), it is necessary to set those fields back to zero after they have been printed and added into the final totals. This process is indicated in column 39. The B entries indicate that the totals, SUB QTY and SUBCOS, are to be blanked out, or zeroed out, after they are printed, when the control field (ITEM) changes. They are set back to zero so that they can correctly accumulate the quantities and costs for the next item.



```
ABCD        25      25.00
            60      60.00
            65      65.00

           150     150.00      SUBTOTALS

MNOP        10        .20
          2000      40.00
            50       1.00
           180       3.60

          2240      44.80      SUBTOTALS

XYZZ        40      80.00
            49      98.00

            89     178.00      SUBTOTALS

          2479     372.80      FINAL TOTALS
```

- **Figure 5-68. Group Printing — Report Showing Subtotals**



File Description Specification

Line (3-5)	Form Type (6)	Filename (7-14)	File Type (15)	File Designation (16)	Record Length (24-27)	Device (40-46)
02	F	GROUPING	I	P	96	MFCU1
03	F	OUTPUT	O		96	PRINTER
04	F					

IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program

Programmer

Date

Punching Instruction: Graphic / Punch

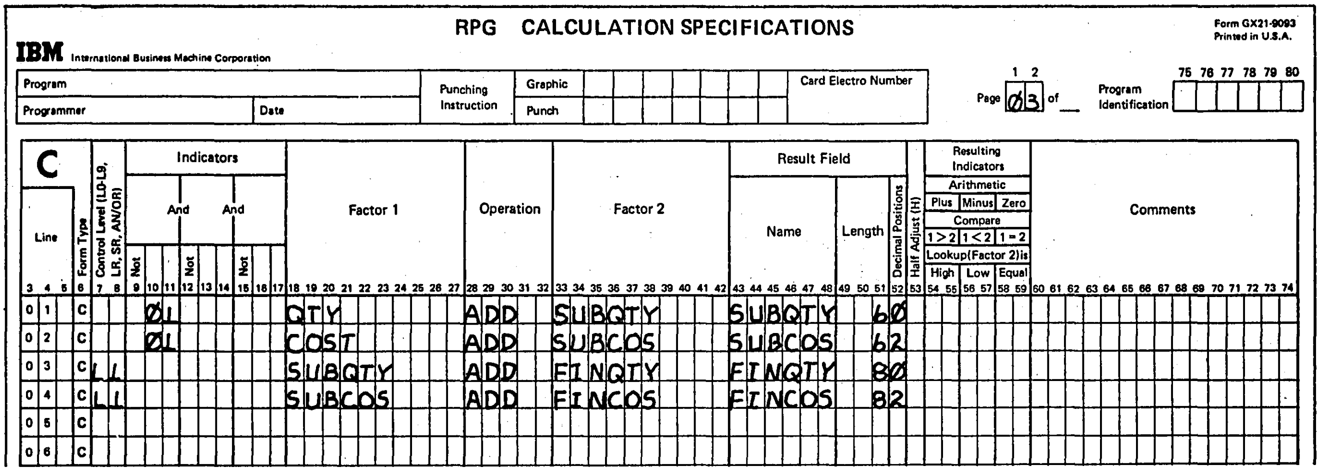
Card Electro Number

Page Ø2 of __

Program Identification (75-80)

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Record Identifying Indicator (19-20)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)
01	I	GROUPING	AA	Ø1					
02	I				1	4		ITEM	L1
03	I				5	8	Ø	QTY	
04	I				9	12	2	COST	
05	I								

- **Figure 5-69 (Part 1 of 2). Group Printing — Printing Subtotals**



IBM International Business Machine Corporation

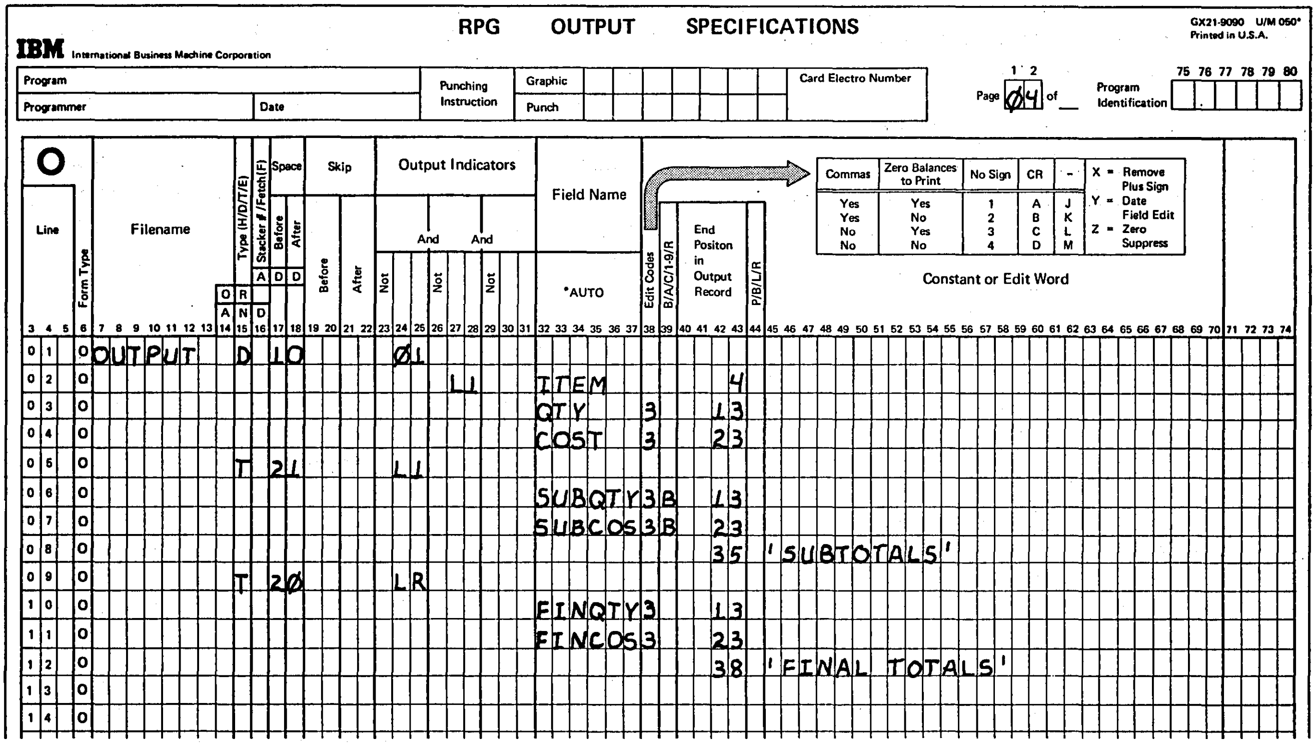
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Programmer | Date

Page Ø3 of ___ Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Indicators (And)	Indicators (And)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C		Ø1			QTY	ADD	SUBQTY	SUBQTY	6	Ø			
02	C		Ø1			COST	ADD	SUBCOS	SUBCOS	6	2			
03	C	L1				SUBQTY	ADD	FINQTY	FINQTY	8	Ø			
04	C	L1				SUBCOS	ADD	FINCOS	FINCOS	8	2			
05	C													
06	C													



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Programmer | Date

Page Ø4 of ___ Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker # / Fetch (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Output Indicators (And)	Output Indicators (And)	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	OUTPUT	D		1	O			Ø1								
02	O									L1		ITEM			4		
03	O											QTY	3		13		
04	O											COST	3		23		
05	O		T		2	1			L1								
06	O											SUBQTY	3	B	13		
07	O											SUBCOS	3	B	23		
08	O														35		'SUBTOTALS'
09	O		T		2	Ø			LR								
10	O											FINQTY	3		13		
11	O											FINCOS	3		23		
12	O														38		'FINAL TOTALS'
13	O																
14	O																

- **Figure 5-69 (Part 2 of 2). Group Printing — Printing Subtotals**

Group Printing with Two Control Level Indicators

Certain programs require two or more control levels. Nine levels (L1-L9) are possible. This program uses two of these, L1 and L2, to produce the report shown in Figure 5-70.

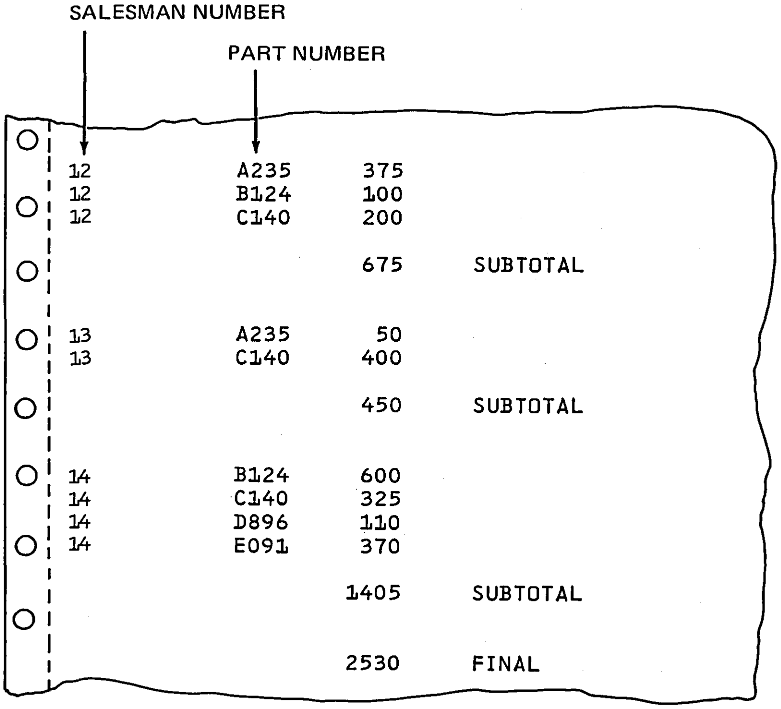
Each record in the input file contains the part number of an item, the quantity of items sold, and a number to identify the salesman who sold those items. The file has been previously arranged by salesman number. That is, all records for salesman number 12 are together, all records for salesman number 13 are together, and so forth. Records are also grouped by item number within each salesman group.

On the printed report, the salesman number is to be printed once; part numbers are in the next column with the sums of quantities for each part number in the rightmost column. Subtotals are printed for each salesman and a final total is printed at the end of the report.

Figure 5-71 shows the coding to produce the report. Since the PARTNO field changes most frequently (there are several part numbers for each salesman) that field is the lowest level control field and is assigned indicator L1 on the Input sheet. The next higher level control field, which does not change as frequently, is SLSNUM; therefore, SLSNUM is assigned indicator L2. The calculation on the first line (01 indicator specified) occurs for every record that is read. QTY is added to a total, QTYSUM. Line 2 occurs only when the control field, PARTNO, changes, because L1 (columns 7 and 8) controls this calculation. So, when L1 is on, QTYSUM is added to another total, SUBTOT. The third line describes the calculation of a final total, FINOT. This calculation occurs when L2 is on (when the control field, SLSNUM, changes).

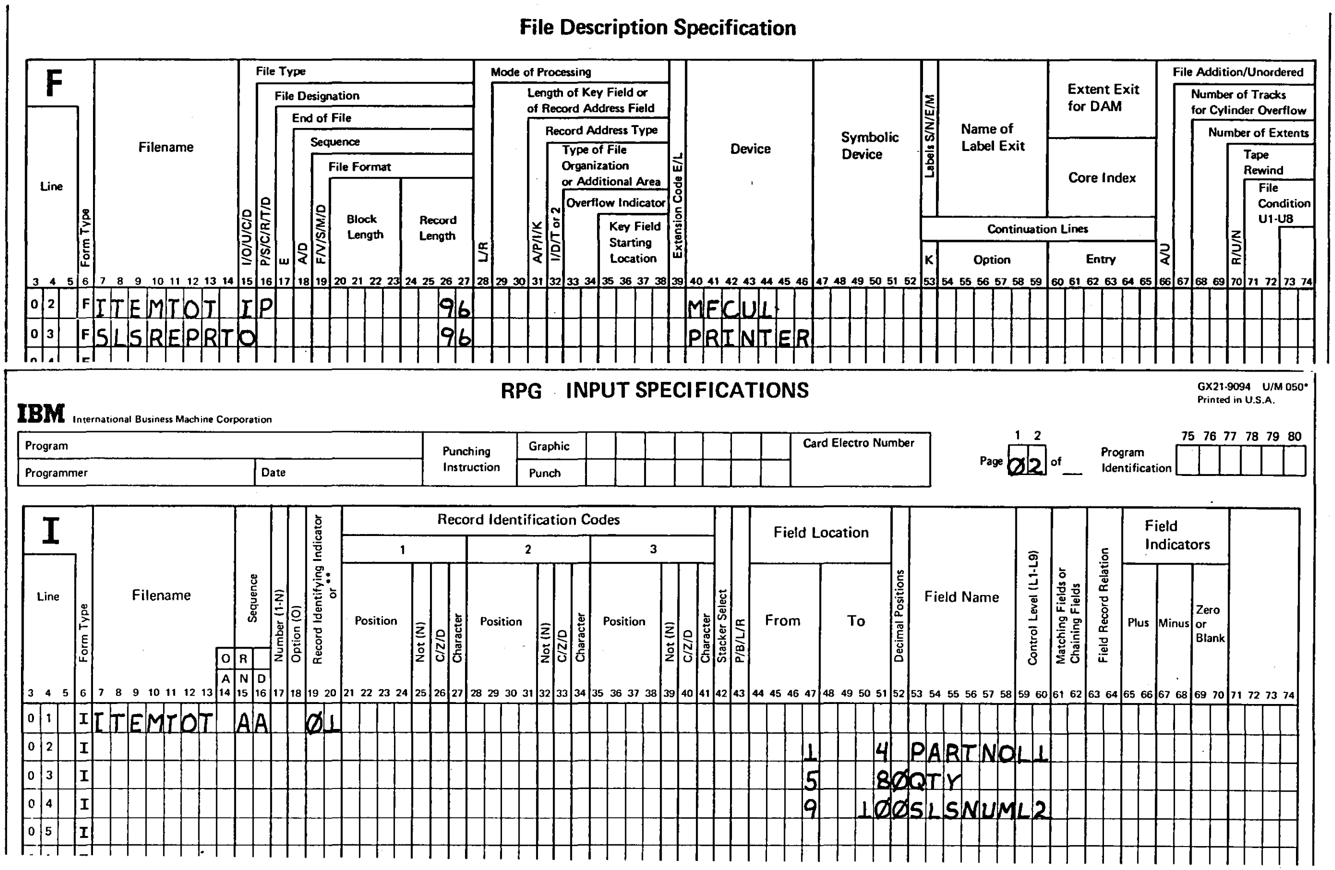
When the higher level control level indicator (L2) is on, the lower level indicator (L1) is also on. As shown on the Output-Format sheet, when L1 is on, PARTNO and QTYSUM are printed (lines 03 and 04). However, the field SLSNUM is conditioned by L2, so it is printed only for the first record of each L2 control group. When L2 is on, SUBTOT and the constant SUBTOTAL are printed (lines 05-07). When LR is on, FINTOT and the constant FINAL are printed.

The two subtotals, QTYSUM and SUBTOT, are zeroed after printing by entering a B in column 39 (lines 04 and 06). Detail lines need never be blanked after, nor does a final total (regulated by the LR indicator).


SALESMAN NUMBER
PART NUMBER

```
12        A235     375
12        B124     100
12        C140     200

                   675     SUBTOTAL

13        A235      50
13        C140     400

                   450     SUBTOTAL

14        B124     600
14        C140     325
14        D896     110
14        E091     370

                  1405     SUBTOTAL

                  2530     FINAL
```

Figure 5-70. Group Printing — Report Produced Using Two Control Level Indicators



File Description Specification

Line	Form Type	Filename	File Type	File Designation	Record Length	Device
02	F	ITEMTOT	I	P	96	MFCU1
03	F	SLSREPRT	O		96	PRINTER
04	F					

RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program		Punching Instruction	Graphic
Programmer	Date		Punch

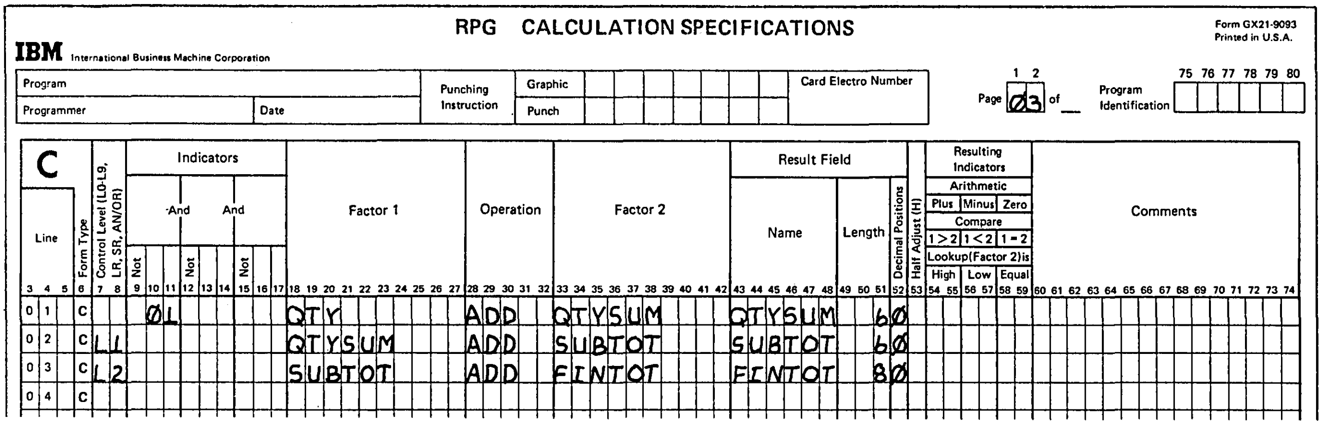
Card Electro Number

Page 02 of ___

Program Identification

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)
01	I	ITEMTOT	AA	01					
02	I				1	4		PARTNO	L1
03	I				5	8	0	QTY	
04	I				9	10	0	SLSNUM	L2
05	I								

Figure 5-71 (Part 1 of 2). Group Printing with Two Control Level Indicators



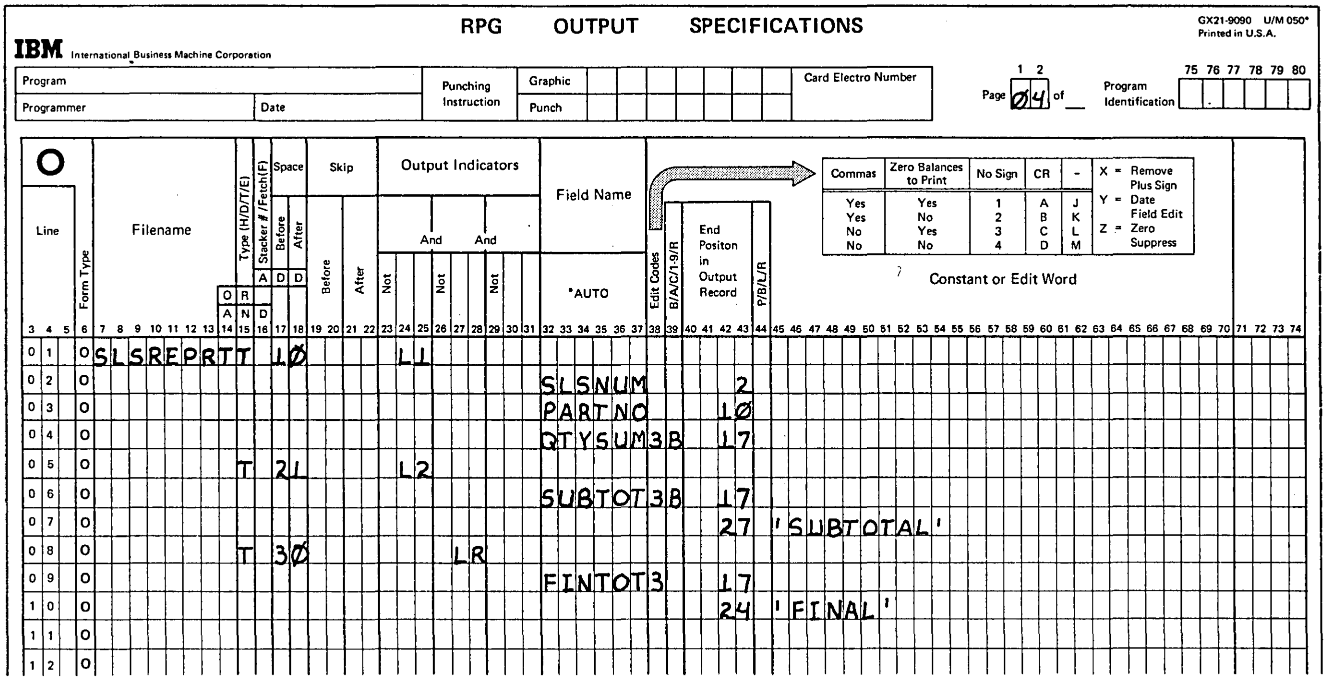
RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page Ø3 of __ | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C		Ø1	QTY	ADD	QTYSUM	QTYSUM	6	Ø			
02	C	L1		QTYSUM	ADD	SUBTOT	SUBTOT	6	Ø			
03	C	L2		SUBTOT	ADD	FINTOT	FINTOT	8	Ø			
04	C											



RPG OUTPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page Ø4 of __ | Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	SLSREPRT	T	1	Ø			L1						
02	O								SLSNUM			2		
03	O								PARTNO			1Ø		
04	O								QTYSUM	3	B	17		
05	O		T	2	1			L2						
06	O								SUBTOT	3	B	17		
07	O											27		'SUBTOTAL'
08	O		T	3	Ø			LR						
09	O								FINTOT	3		17		
10	O											24		'FINAL'
11	O													
12	O													

Figure 5-71 (Part 2 of 2). Group Printing with Two Control Level Indicators

BINARY FIELD OPERATIONS (CONTROLLING SWITCHES)

Note: This topic is intended for IBM System/3 Model 6, Model 10, Model 12, and Model 15 programmers.

RPG II provides certain operation codes which set and test individual bits in storage. These individual bits can be set and tested to allow you further control over processing. When this is done, the bits are called switches and their functions are similar to that of RPG II indicators. The operation codes which set and test the bits are known as *binary field operations.* A binary field is a one-byte field containing 8 bits identified left to right by the digits 0-7. The bits can be set on, set off, and tested. Since each bit can be utilized, there are eight indicators in every byte.

When using binary field operations, remember how data fields are initialized by the system:

- Alphameric fields are initialized to Hexadecimal '40'.
- Numeric fields are initialized to Hexadecimal 'F0'.

You should initialize the binary field containing the bits to be set and tested to binary zero (Hexadecimal '00') at the beginning of the program.

BITON Operation Code

Figure 5-72 shows a Calculation sheet containing the BITON operation code. This operation code causes specified bits in Factor 2 to turn on (set to 1) in the field named in the Result Field. The field named in the Result Field must be a one-position alphameric field. Since it is one position in length, a 1 must be entered in column 51 of Field Length. One or more of the eight bits can be turned on. To turn on the first bit in a field, enter 0 in Factor 2. These bit numbers must be enclosed by apostrophes.

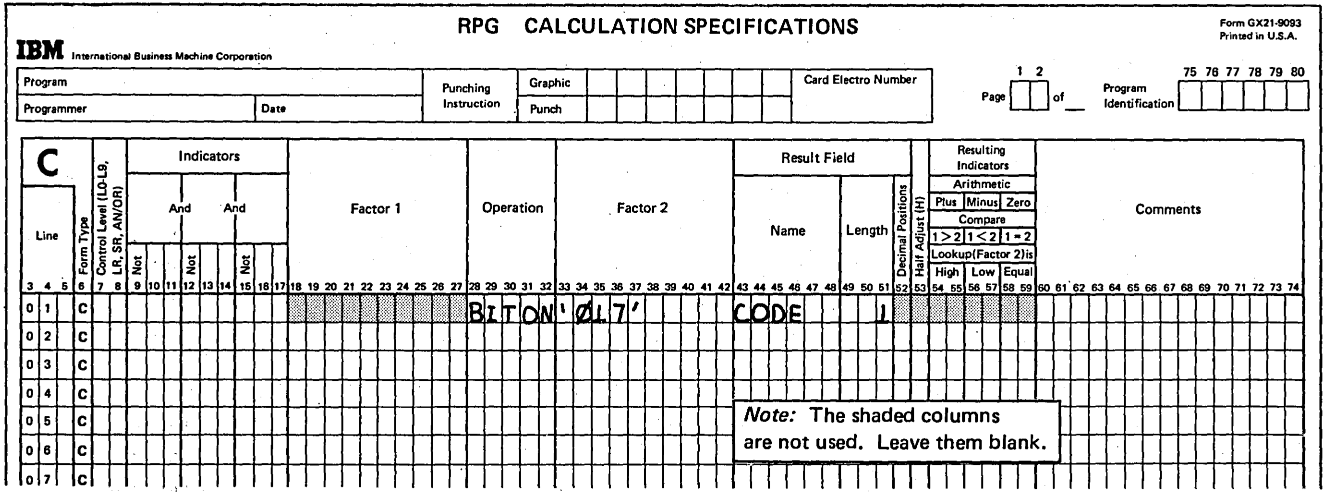
You can condition the operation with indicators in columns 7-17. You may also turn on a bit in an array element, but that array element must be one position in length.

In Figure 5-72, bits 0, 1, and 7 are set to 1 in the binary field labeled CODE.

BITOF Operation Code

Figure 5-73 is a sample Calculation sheet containing the BITOF operation code. This operation code causes specified bits identified in Factor 2 to turn off (set to 0) in a field named in the Result Field. In Figure 5-73, bits 0, 3, and 4 are turned off (set to 0) in the binary field labeled CODE.

All other specifications are the same as those specified under *BITON Operation Code.*



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page (1 2) of
Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C			▨	BITON	'Ø17'	CODE	1	▨	▨	▨	
02	C											
03	C											
04	C											
05	C											
06	C											
07	C											

Note: The shaded columns are not used. Leave them blank.

Figure 5-72. The BITON Operation Code

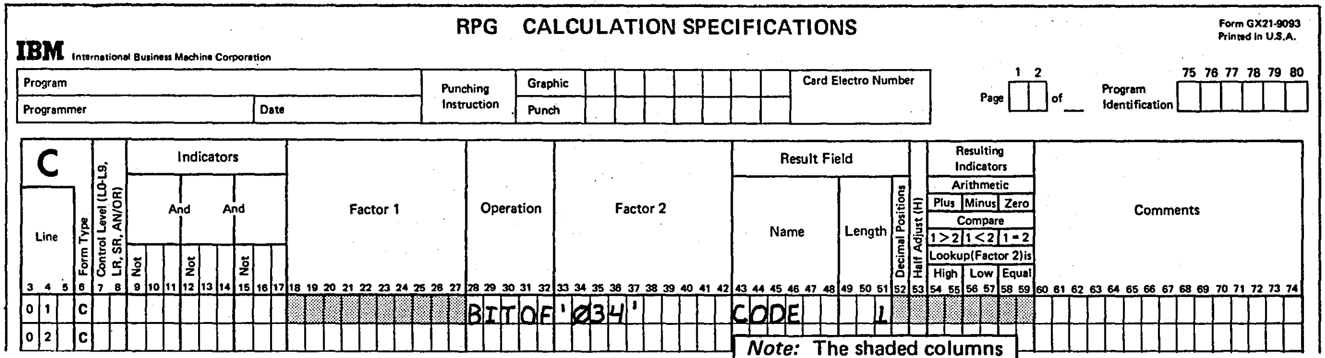
TESTB Operation Code

Figure 5-74 is a sample Calculation sheet with the TESTB operation code. This operation code causes specified bits identified in Factor 2 to be tested for an off or on condition. Resulting indicators in columns 54-59 are set depending upon the conditions. At least one resulting indicator must be used with the TESTB operation, and as many as three can be named for one operation. Two indicators may be the same for one TESTB operation, but not three. Resulting indicators in these columns have the following meanings:

- Columns 54-55: An indicator in these columns is turned on if all the bits in Factor 2 are off (set to 0).
- Columns 56-57: An indicator in these columns is turned on if two or more bits were tested and found to be of mixed status, some bits on and other bits off.
- Columns 58-59: An indicator in these columns is turned on if all the bits in Factor 2 are on (set to 1).

In Figure 5-74, bits 4, 5, and 6 in the binary field named CODE are tested. Resulting indicator 66 is turned on if bits 4, 5, and 6 are off. If some are on and others off, resulting indicator 77 is turned on. If they are all on, resulting indicator 88 is turned on.

All other specifications are the same as those specified under *BITON Operation Code.* However, you need not define the Result Field as one position in length, since this is done when the field is used in a BITON or BITOF operation code.



IBM International Business Machine Corporation

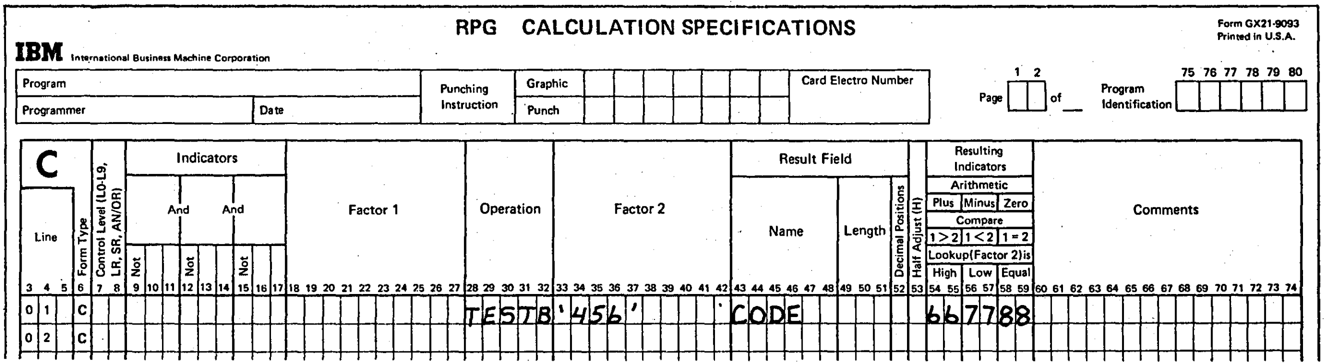
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	Resulting Indicators High	Low	Equal	Comments
01	C				BITOF	'∅34'	CODE	1						
02	C													

Note: The shaded columns are not used. Leave them blank.

Figure 5-73. The BITOF Operation Code



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	Resulting Indicators High	Low	Equal	Comments
01	C				TESTB	'456'	CODE				66	77	88	
02	C													

Figure 5-74. The TESTB Operation Code

Example

Fields are sometimes present in customer master files to indicate particular types of customers. When such a master file is created, each of the conditions indicating a particular customer type is represented in a record by a one-position field. Since each position occupies one byte of storage, four positions indicating customer types will be stored in four bytes of storage. You can use binary field operations to convert each one-byte record position to one bit of information on disk. Therefore, four bytes of information can be reduced to four bits of information on disk.

For example, assume you have a customer master file on cards. You have four columns containing the following information:

- Whether the customer is a wholesaler or retailer.
- If the customer is entitled to a discount.
- If orders should be checked by the credit department.
- If due to a bad payment history, the shipment should be sent cash on delivery.

Now you want to place the card file on disk, and the information from the four columns in four bits in a binary field labeled CHECK. The four columns will be labeled WHLSE, DSCT, CREDIT, and COD respectively. The folowing operations should be performed:

1. If WHLSE is equal to 1, turn on bit 0 in CHECK.
2. If DSCT is equal to 1, turn on bit 1 in CHECK.
3. If CREDIT is equal to 1, turn on bit 2 in CHECK.
4. If COD is equal to 1, turn on bit 3 in CHECK.

Figure 5-75 shows correct coding for this problem. Remember that before setting up data in a binary field, the binary field should be set to binary zero. This can be done by the BITOF instruction (Line 1, Figure 5-75).

INCREASING THE SPEED OF OPERATIONS (DUAL I/O AREAS)

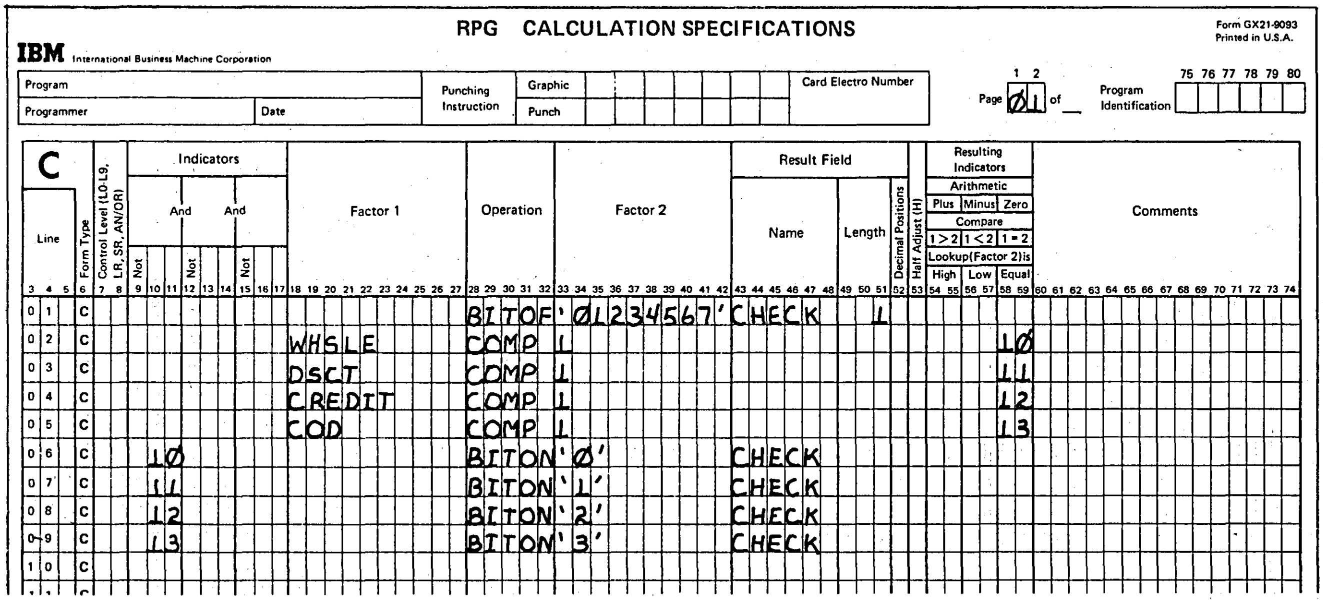
During the normal RPG II cycle, a record is read, calculations are performed, and output is produced. The cycle is repeated for each record.

The speed at which the cycle is done depends upon the speed at which records are read and output produced. Calculations take less time than reading, printing, writing to disk, or punching. The speed of doing output can be increased by using dual input/output areas.

Dual Input Areas

When dual input areas are used, the program cycle is changed. First a record is read. At the same time, calculations are being performed on this record, another record is being read. Thus, the contents of two records are in the computer at the same time. Figure 5-76 shows how the records are processed when two input areas are used.

Dual input areas can be specified for sequential disk files or direct disk files processed sequentially, for card files designated as input files, or for tape input files. No stacker selection can be specified for card files. Dual input areas cannot be specified for combined or update files, table, or demand files. When shared input/output is specified in the header card, all devices which can use shared input/output are automatically excluded from use of dual input areas. *(Note to Model 6 Programmers:* Dual input areas can be used for data recorder input files. Dual input areas cannot be assigned to disk, data recorder, or ledger card device files using a shared input/output area, specified in column 48 of the Control sheet.)



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page Ø1 of | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Plus (High, 1>2)	Minus (Low, 1<2)	Zero (Equal, 1=2)	Comments
01	C				BITOF	'Ø1234567'	CHECK	1						
02	C			WHSLE	COMP	1							1Ø	
03	C			DSCT	COMP	1							11	
04	C			CREDIT	COMP	1							12	
05	C			COD	COMP	1							13	
06	C		1Ø		BITON	'Ø'	CHECK							
07	C		11		BITON	'1'	CHECK							
08	C		12		BITON	'2'	CHECK							
09	C		13		BITON	'3'	CHECK							
10	C													

Figure 5-75. Example of Binary Field Operations

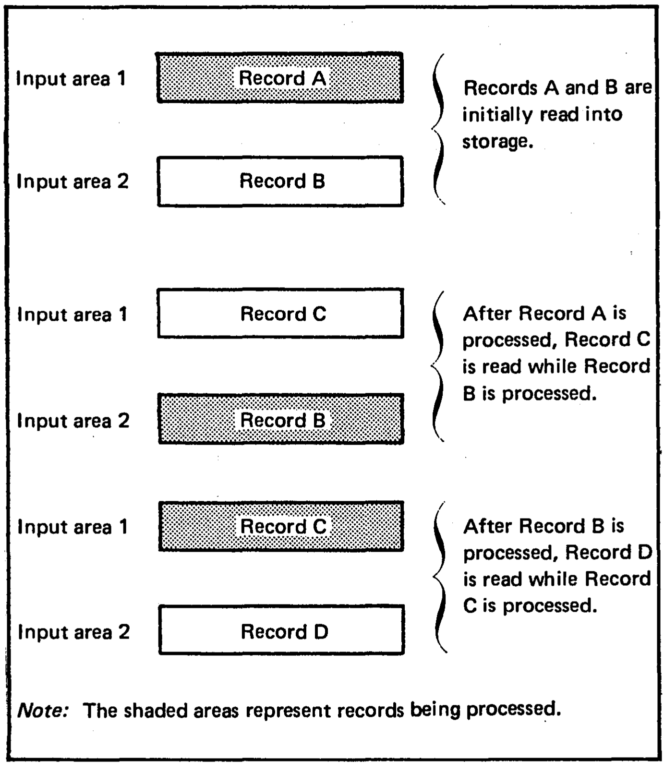
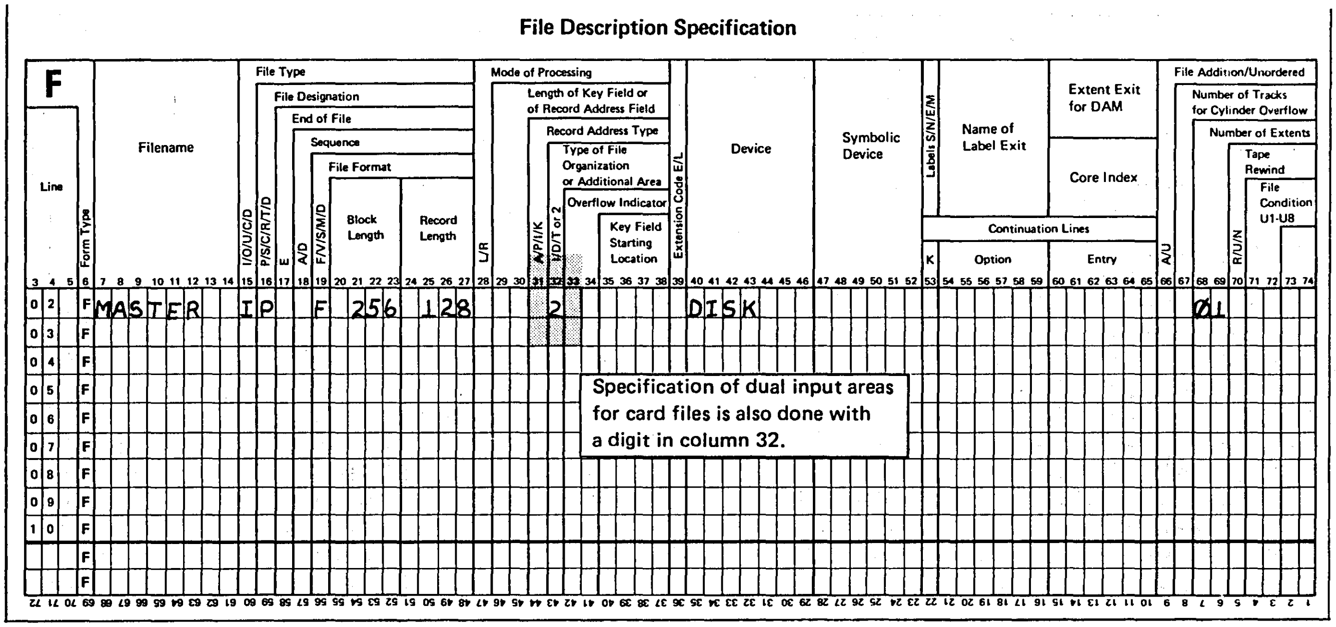

Input area 1
Record A
Input area 2
Record B
Records A and B are initially read into storage.
Input area 1
Record C
Input area 2
Record B
After Record A is processed, Record C is read while Record B is processed.
Input area 1
Record C
Input area 2
Record D
After Record B is processed, Record D is read while Record C is processed.
Note: The shaded areas represent records being processed.

Figure 5-76. Dual Input Areas

Dual input areas require more computer storage space than one input area, because two records are in storage during each cycle. If you have a large program, you might not have enough storage space to accommodate two input areas. The effect of dual input areas can be determined only if you have knowledge of a program's processing requirements and experience in RPG II programming. In some cases, you can only make a final determination by actual experimentation. *(Note to Disk Programmers:* If your program plus two input areas require more space than is available, certain RPG II object cycle routines remain on disk during execution and are called into storage as needed. If too many routines remain on disk, the performance of your program may be decreased by the use of dual input areas rather than increased.)

Specifications

One entry on the File Description sheet is required to specify dual input areas: any digit (1-9) in column 32 assigns dual input areas for the specified file. Figure 5-77 shows the file MASTER has been assigned dual input areas.



File Description Specification

```
0 2 F MASTER   IP  F 256 128    2     DISK                Ø1
```

Specification of dual input areas for card files is also done with a digit in column 32.

Figure 5-77. Specifying a Dual Input Area

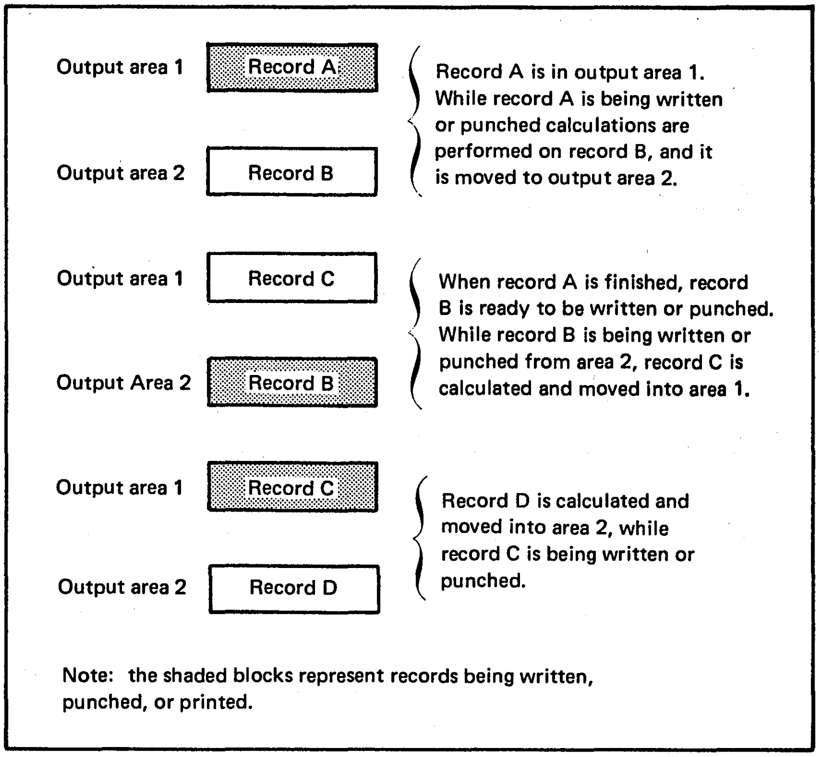
Dual Output Areas

When dual output areas are used, the program cycle is changed. A record is written or punched out at the same time calculation and output operations are being done internally to produce the next record. (Calculation operations are not done at the same time as writing or punching when only one output area is used.) Figure 5-78 shows how output records are produced using dual output areas.

Dual output areas, like dual input areas, require more computer storage. Consequently, the same space considerations that apply to dual input areas also apply to dual output areas. Dual output areas can be used for sequential and direct disk output files, Model 10 printer files (5203 printer), tape files, and card output files. Dual output areas cannot be specified for combined or update files. Also, dual output areas cannot be assigned to Model 6 data recorder, disk or ledger card device files and Model 10 or Model 15 disk files that use a shared input/output area.

Specifications

One entry is required on the File Description sheet to specify dual output areas: any digit (1-9) can be entered in column 32 for an output file. Figure 5-79 shows the file PRINT has been assigned dual output areas.



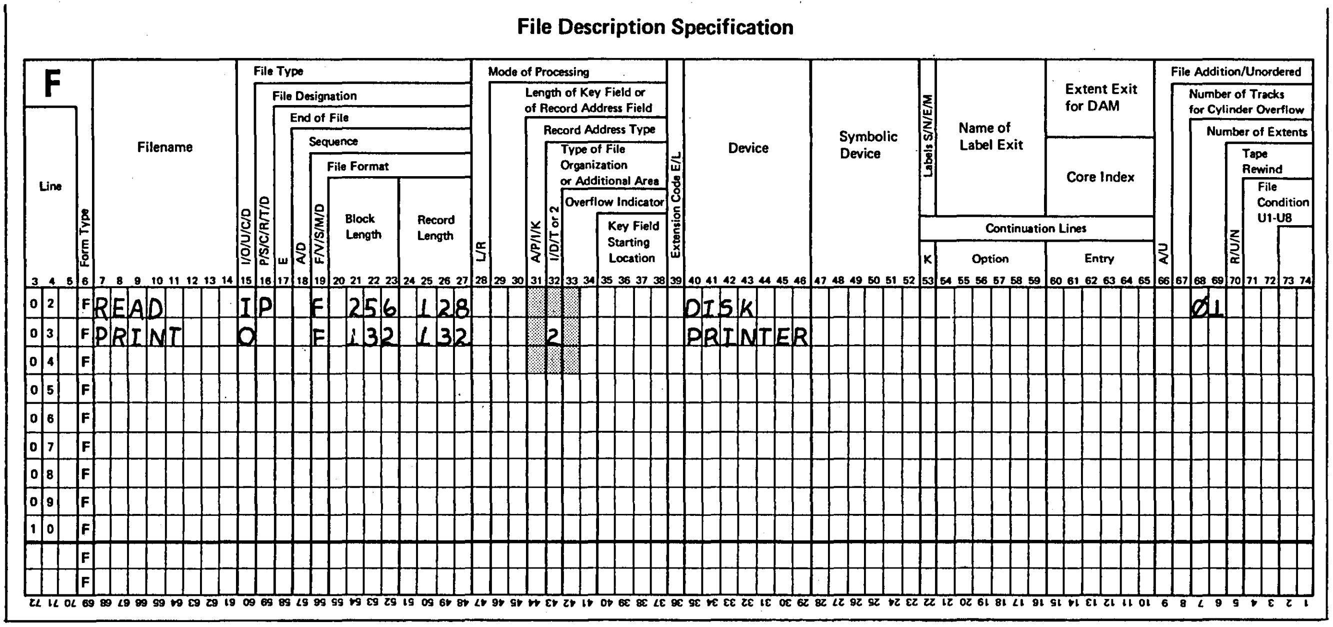
Record A is in output area 1. While record A is being written or punched calculations are performed on record B, and it is moved to output area 2.

When record A is finished, record B is ready to be written or punched. While record B is being written or punched from area 2, record C is calculated and moved into area 1.

Record D is calculated and moved into area 2, while record C is being written or punched.

Note: the shaded blocks represent records being written, punched, or printed.

Figure 5-78. Dual Output Areas



File Description Specification

F

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V/S/M/D	Block Length	Record Length	L/R	Length of Key Field or of Record Address Field	A/P/I/K	I/D/T or 2	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind R/U/N	File Condition U1-U8
02	F	READ	I	P			F	256	128								DISK							Ø1		
03	F	PRINT	O				F	132	132				2				PRINTER									
04	F																									
05	F																									
06	F																									
07	F																									
08	F																									
09	F																									
10	F																									
	F																									
	F																									

Figure 5-79. Specifying Dual Output Areas

Additional Uses of Indicators

1. What effect do halt indicators have on System/3 operations? How can you use halt indicators to handle error situations?

2. Describe a method to eliminate an output file for one program run without having to rewrite specifications and recompile the program.

3. When conditioning a calculation specification with an indicator, what happens when the indicator is on? What happens when it is off?

4. Describe what the TESTZ operation does.

5.

Factor 1	Operation	Factor 2	Result Field Name	Result Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Arithmetic: Plus Compare: 1 > 2 Lookup (Factor 2) is: High	Resulting Indicators Arithmetic: Minus Compare: 1 < 2 Lookup (Factor 2) is: Low	Resulting Indicators Arithmetic: Zero Compare: 1 = 2 Lookup (Factor 2) is: Equal
18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51	52	53	54 55	56 57	58 59
	TESTZ		TEST				93	94	95

Using this TESTZ specification, tell what the status of indicators 93, 94, and 95 will be when the field TEST contains:

a. ABC
b. &/*
c. JOH
d. 789

6.

Factor 1	Operation	Factor 2	Result Field Name	Result Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Arithmetic: Plus Compare: 1 > 2 Lookup (Factor 2) is: High	Resulting Indicators Arithmetic: Minus Compare: 1 < 2 Lookup (Factor 2) is: Low	Resulting Indicators Arithmetic: Zero Compare: 1 = 2 Lookup (Factor 2) is: Equal
18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51	52	53	54 55	56 57	58 59
FIRST	COMP	SECOND					96	97	98

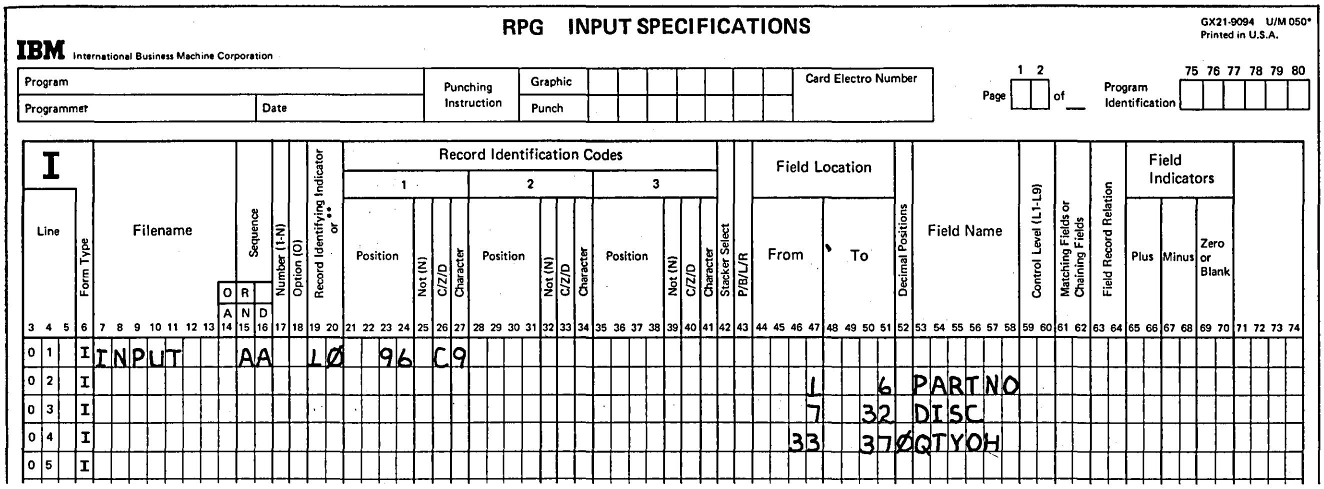
Using this COMP specification, tell what the status of indicators 96, 97, and 98 will be when the fields FIRST and SECOND contain:

	First	*Second*
a.	16942	17942
b.	16000	15000
c.	19645	19645
d.	18921	18931

Controlling Operations on the Basis of the Next Record in a File

7. Basically, what does the look ahead facility allow you to do? What limitations apply to its use?

8. To the input specifications given add those which will allow you to look ahead in order to read the next part number (PARTNO) and next code in column 96.



IBM International Business Machines Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program ____ Programmer ____ Date ____ Punching Instruction: Graphic ____ Punch ____ Card Electro Number ____ Page ☐☐ of ___ Program Identification ☐☐☐☐☐☐

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P/B/L/R (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	(71-74)
01	I	INPUT	AA			LØ	96		C	9																					
02	I																				1	6		PARTNO							
03	I																				7	32		DISC							
04	I																				33	37	Ø	QTYOH							
05	I																														

Moving Data

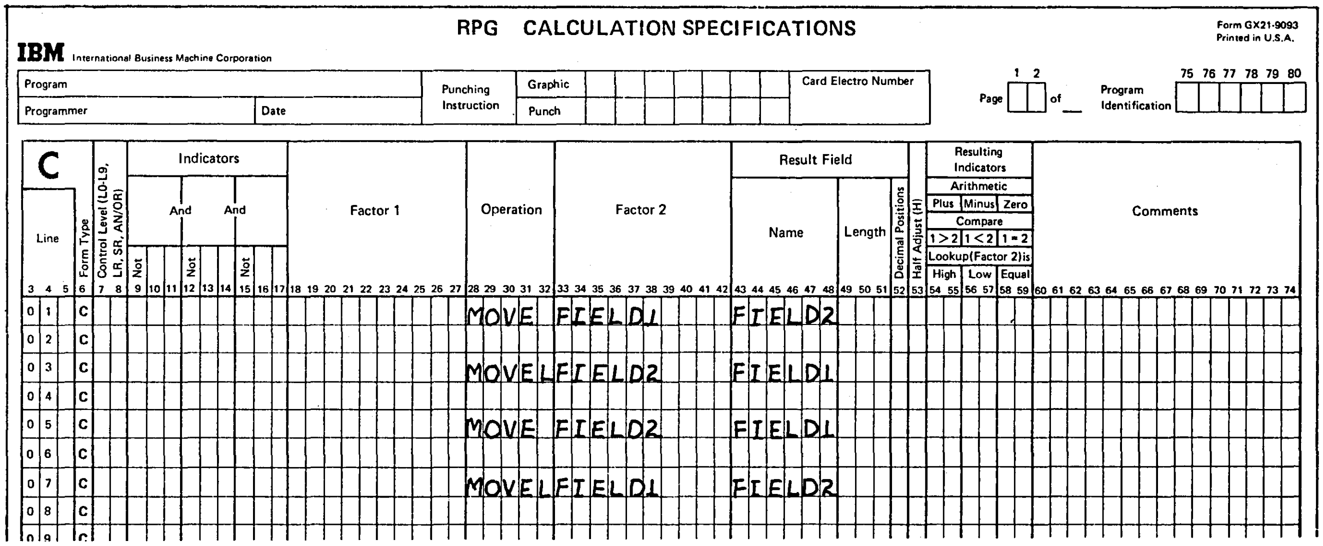
9. With MOVE, which position of Factor 2 is moved first and where is it positioned in the Result Field?

10. What happens in a MOVE operation if Factor 2 is larger than the Result Field? What happens if it is smaller?

11. With MOVEL, which position of Factor 2 is moved first and where is it positioned in the Result Field?

12. What sign does a numeric Result Field have after a MOVEL operation?

13. FIELD1, a 4-position positive numeric field, contains 3456; FIELD2, an 8-position negative numeric field, contains 87654321. What will the contents of the Result Field be after each of the following instructions is executed, if FIELD1 and FIELD2 contain the above values before each instruction is executed?



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C				MOVE	FIELD1	FIELD2					
02	C											
03	C				MOVEL	FIELD2	FIELD1					
04	C											
05	C				MOVE	FIELD2	FIELD1					
06	C											
07	C				MOVEL	FIELD1	FIELD2					
08	C											
09	C											

Branching in Calculations

14. What is the purpose of branching?

15. How is branching specified in RPG II?

16. By using branching, how can you structure a program, which has several record types each of which requires different calculation operations, so that it is easy to write and efficient to run?

17. What must you always include in a program that has a loop? What will happen if you do not include this?

Using Subroutines in Calculations

18. When should a subroutine be used?

19. What are the operations used to define and execute a subroutine? What entry must be made in each calculation line of a subroutine that is different from all other calculations?

20. What limitations in the use of GOTO and TAG apply to subroutines?

21. Where must subroutines be coded?

Special Uses of Control Level Indicators

22. When would you use the L0 indicator?

23. How can you specify calculations to be performed only on the first card of a group?

24. When should you specify blank after (B in column 39 of the Output-Format sheet) for an output field?

Binary Field Operations

25. What are bit switches used for?

26. Code the calculation specification to:
    a. Set on bits 4 and 7 in a field called TESTER.
    b. Set off bits 1, 2, and 3 in TESTER.
    c. Test to see whether bits 1, 2, and 3 in TESTER are all on or all off. Set on indicator 01 if they are all on and set on indicator 02 if they are all off.

Dual Input/Output Areas

27. For which device and file types can you specify dual input/output areas on your system?

Answers To Review 5

1. Halt indicators may be turned on as record identifying indicators, as a result of a test on a field, or as a result of calculations. When they are on, they cause the system to halt after all calculations and output operations have been performed for that program cycle.

   Because a program cycle is completed before the system halts, operations may be performed on erroneous data. Halt indicators, when used as conditioning indicators, allow you to bypass calculations and output operations which would usually be done. They also allow you to print error messages and stacker select any cards with erroneous information.

2. Use an external indicator (U1-U8) to condition the output file and operations specified for that file. When the indicator is on the file is used; when it is off, the file is not used. The file is conditioned by an external indicator specified in columns 71-72 of the File Description sheet. The output specifications must be conditioned by that same external indicator entered in columns 23-31 of the Output-Format sheet.

3. If the indicator is on, the step will be executed. If the indicator is off, the step will not be executed.

4. TESTZ examines the zone portion of the leftmost character in an alphameric field. If that position contains & or A-I, the plus indicator will turn on. If that position contains -, { , or J-R, the minus indicator turns on. For all other characters, the zero indicator turns on.

5.

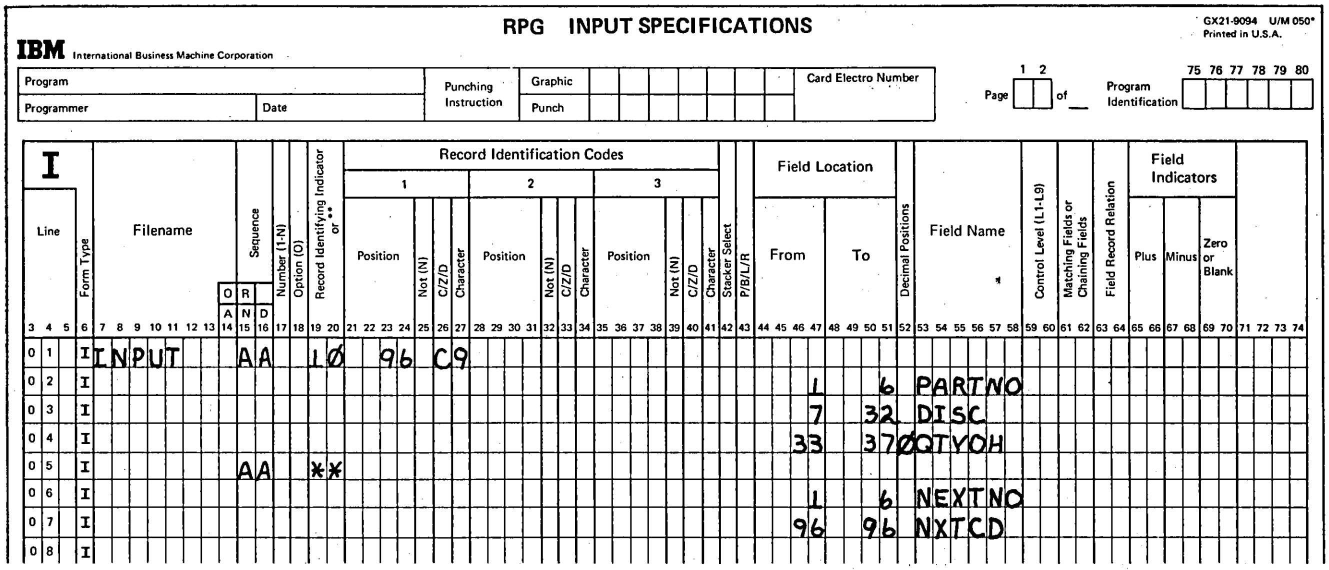
	93	*94*	*95*
a.	on	off	off
b.	on	off	off
c.	off	on	off
d.	off	off	on

6.

	96	*97*	*98*
a.	off	on	off
b.	on	off	off
c.	off	off	on
d.	off	on	off

7. Look ahead allows you to use data on the next record to be processed. Normally only the data on the record currently being processed is available to the RPG II program. Look ahead is most often used with input files. If it is used with combined or update files, information in the look ahead field while the combined or update file is being processed will be from the record currently being processed, not from the next record in the file.

8. ** must be specified in columns 19-20 to indicate that the fields listed are to be looked at in the next card available for processing. Look ahead fields must be given different field names than those used when describing the file. Any alphabetic characters may be used in columns 15-16.



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program		Punching Instruction	Graphic		Card Electro Number	Page □□ of ___
Programmer	Date		Punch			Program Identification □□□□□□

Line	Form Type	Filename	Sequence (OR / AND)	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank
01	I	INPUT	AA			1Ø	96		C	9																				
02	I																				1	6		PARTNO						
03	I																				7	32		DISC						
04	I																				33	37	Ø	QTYOH						
05	I		AA			**																								
06	I																				1	6		NEXTNO						
07	I																				96	96		NXTCD						
08	I																													

9. The righthand, low-order position of Factor 2 is moved first, to the righthand, low-order position of the Result Field.

10. If Factor 2 is larger, the number of positions moved will be equal to the size of the Result Field. If Factor 2 is smaller, the whole field will be moved and the lefthand positions of the Result Field will be unchanged.

11. With MOVEL, the leftmost position of Factor 2 is moved first to the leftmost position of the Result Field.

12. The sign of the Result Field after a MOVEL operation is performed will be the same as that of the low-order character of Factor 2 unless Factor 2 is smaller than the Result Field in which case the sign is unchanged.

13. a. $8765345\overset{+}{6}$
    b. $876\bar{5}$
    c. $432\bar{1}$
    d. $3456432\bar{1}$

14. Branching alters the sequence in which calculations are executed. It allows you to:
    a. Skip calculation operations.
    b. Execute operations in an order other than the normal sequential order.
    c. Perform the same calculations several times in one cycle.

15. Branching in RPG II is specified by the operations GOTO and TAG. GOTO tells the computer to branch to a location indicated in Factor 2 of the instruction. The TAG statement is placed at the point in your program to which you want to branch. The name in Factor 1 of the TAG statement is the same as the name in Factor 2 of the appropriate GOTO. Every GOTO requires a TAG or the program will not know where to branch to. Several GOTO statements may branch to the same TAG, but the name on each TAG statement must be unique.

16. Condition several GOTO statements with the various record type indicators. Write the set of specifications for each record type separately and include them in the program. The set of specifications for each record type will begin with a TAG statement and end with a GOTO statement which branches to the end of all calculations. The program will test the record type and branch to the correct set of specifications. At the end of the set of specifications, it will branch around all other specifications to the end of the calculation section.

17. If a program has a loop, there must be some way to stop the looping. The branch back instruction must be conditioned so that when certain conditions have been met, the statement will not be executed. If this condition is not specified or cannot be met, the program will go into an endless loop.

18. A subroutine can be used whenever the same calculations must be executed at several different places in a program.

19. The first line of a subroutine must have the BEGSR operation code in columns 28-32 with the subroutine name in Factor 1. The last line in a subroutine must have ENDSR operation code in columns 28-32. This line can have a name in Factor 1, and this name can then be referenced by a GOTO statement. Every subroutine line must have SR in columns 7-8.

20. No branches can be made from a GOTO statement within a subroutine to a TAG statement outside that subroutine. No branches can be made from outside the subroutine to a TAG statement within the subroutine.

21. All subroutines must appear on the Calculation sheet after all detail and total calculations.

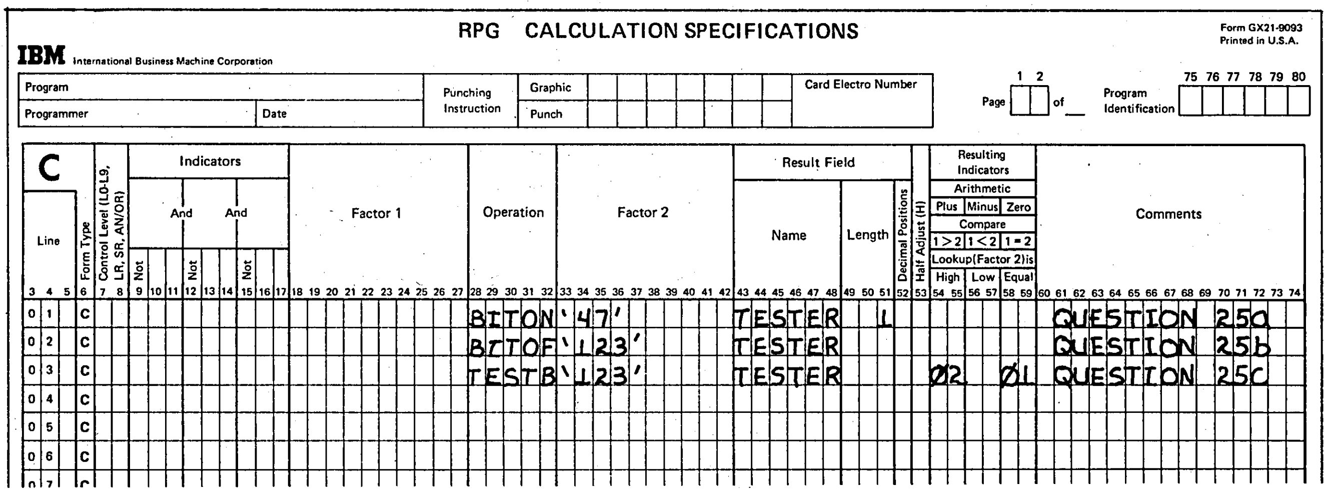
22. L0 would be used if you need to perform some calculation steps at total calculation time and there is no normal control level indicator (L1-L9) available. A common use of this is to set on the other control level indicators when control fields are not available.

23. Detail calculations are distinguished from total calculations by the fact that columns 7-8 of the Calculation sheet are left blank for detail calculations. Since the control level indicator stays on through detail calculations it can be used like any other indicator in columns 9-17 to control detail calculations. The control level indicator will be on for only the first card of a new control group.

24. Blank after should be used to reset a field to zero that is used to accumulate and print a total for each control group. This allows the field to start with a zero value for the new control group.

25. Bit switches are used to code and test for specified situations. With System/3 Model 10 Disk System and Model 6, they are stored in one-byte alphameric fields in storage and on disk. One example is credit information in an accounts receivable file. The first bit might mean a COD only; the second, payment due in 30 days; the third, credit limit $1000; etc. When these conditions are coded as bit switches they take up less disk space than single character codes that might be used in the same way.

26. See coding sheet.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program	Punching Instruction	Graphic	Card Electro Number	Page (1 2) of ___
Programmer / Date		Punch		Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not (9-11)	And: Not (12-14)	And: Not (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field: Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / 1>2 / High (54-55)	Minus / 1<2 / Low (56-57)	Zero / 1=2 / Equal (58-59)	Comments (60-74)
01	C						BITON	'47'	TESTER	1						QUESTION 25a
02	C						BITOF	'123'	TESTER							QUESTION 25b
03	C						TESTB	'123'	TESTER				Ø2		Ø1	QUESTION 25c
04	C															
05	C															
06	C															
07	C															

27. *Model 10 Card System:*

- MFCU input files (no stacker select; no table or demand files)
- MFCU output files (no stacker select)
- PRINTER output files
- PRINTR2 output files
- TAPE input file
- TAPE output file

Model 10 Disk System and Model 12:

- MFCU or 1442 input files (no stacker select; no table or demand files)
- MFCU or 1442 output files (no stacker select)
- PRINTER output files
- PRINTR2 output files
- DISK input files (sequential and direct only)
- DISK output files (sequential and direct only)
- TAPE input files
- TAPE output files

Model 6:

- DISK input files (sequential and direct only; no shared I/O)
- DISK output files (sequential and direct only; no shared I/O)
- DATA96 input files

Model 15: Same as Model 10 Disk System, plus:

- 2501 input files
- MFCM input files (no stacker select; no table or demand files)
- MFCM output files (no stacker select)

Chapter 6. Match Fields and Multifile Processing

CHAPTER 6 DESCRIBES:

How to assign match fields to one or more files.

Use of match fields to sequence check records in a file.

Using matching records to control the processing of multiple input files.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

How to identify input record types on the Input sheet.

How to use field-record relation.

How to specify stacker selection on the MFCU.

RPG II object program cycle.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Checking the sequence of records within a file using match fields.

Using match fields with field-record relation for more than one record type in a file.

Matching records when there is only one record type in a file.

RPG II logic for processing files by matching records.

Matching records when there are more than one record type in a file.

Matching records when all of the records in one of the files have been processed.

Using match fields and control fields in the same file.

How to determine whether a file should be primary or secondary.

Note: You can use the review questions contained in *Review 6* at the end of this chapter to test your comprehension of the topics in the chapter. Answers follow the review questions.

INTRODUCTION

Match fields are data fields on separate records that are compared to establish a relationship between the records. Match fields have two functions, depending on whether the related records are the same input type file or in separate input type files. (An input type file can be an input, update, or combined file.)

Within a file, match fields are used to *check the sequence of records* on which they appear. The sequence checking is accomplished by comparing match fields in one record with the match fields in the next record *of the same file.* When two or more input type files are used by a program, the sequence of each file can be checked just as sequence was checked for a single file. All files that are sequence checked by match fields must be in the same sequence, either ascending or descending.

In addition to sequence checking records within a file, match fields can also be used to establish a relationship between records that are in separate files. That is, match fields can be used to *match records from two or more input type* files to determine which record is to be processed on the current cycle. If two files are used in the same job and you do not specify the order of processing, the primary file is completely processed first. Only then are records from secondary files processed. By specifying that the order of processing is to be determined by comparing the contents of match fields, however, you can cause records from secondary files that are related to a record in the primary file to be processed before the next primary record.

The processing of more than one input type file, with or without match fields, is termed *multifile processing.* Selection of records from more than one file based on the contents of match fields is known as multifile processing by *matching records.*

Multifile processing is commonly used in applications where data files are set up to contain only a certain type of information. For example, a master payroll file might contain data which does not change often, such as an employee's pay rate. Another payroll file, which would consist of new records every week, might contain data such as the number of hours an employee worked in the week. To produce a paycheck or other report, information from both files must be used. Furthermore, the records from the two files must be processed in a particular order. Matching records can be used both to determine which record to process on each program cycle and to sequence check the records within each file.

Note to Disk Programmers: For ease of depicting input records and files, card-like records are shown in illustrations throughout this chapter. This does not imply that the processing must be done using card files. The files can be read from any System/3 device that can be used as an input device. Also, throughout the chapter, two input-type files are used in examples to illustrate RPG II logic for matching records. RPG II logic and the rules for record selection do not change when more than two files are used. See the RPG II reference manual for your system for examples using more than two files.

CHECKING SEQUENCE OF RECORDS WITHIN A FILE

File Containing Only One Record Type

As you know, an A or D entry made in column 18 of a file description specification indicates that the records in the file described are to be in sequence. If you specify that the file EMPLOYEE is to be in ascending (A) sequence, which employee records in Figure 6-1 are in the correct order? Actually all three arrangements show the file in ascending order: the first is sequenced by department number, the second by name, and the third group by employee number.

As you can see then, before the program can check the sequence of records in a file, you must specify the field or fields which are to determine the order. The fields on which the sequence is to be checked (called match fields) are identified on the Input sheet by coding one of the entries M1-M9 in columns 61-62 (see Figure 6-2).

Records within a file may be sequenced on the basis of one or more data fields. Up to nine fields may be used by assigning one of the entries M1-M9 to each match field. Entering M1 on the same line as you describe the DEPT field (Figure 6-2) causes the records to be sequence checked according to department number. Thus, this file should be in ascending order as shown in the first group of Figure 6-1.

When you specify more than one match field to check sequence, the program considers all the match fields to be one continuous field, even though the fields may not be adjacent in a record. For this reason, all match fields assigned to a particular record type are considered to have the same type of data (alphameric or numeric). The individual fields are checked in order according to the level of the match field entry assigned to the field. M1 is the lowest level; M9 is considered the most important.

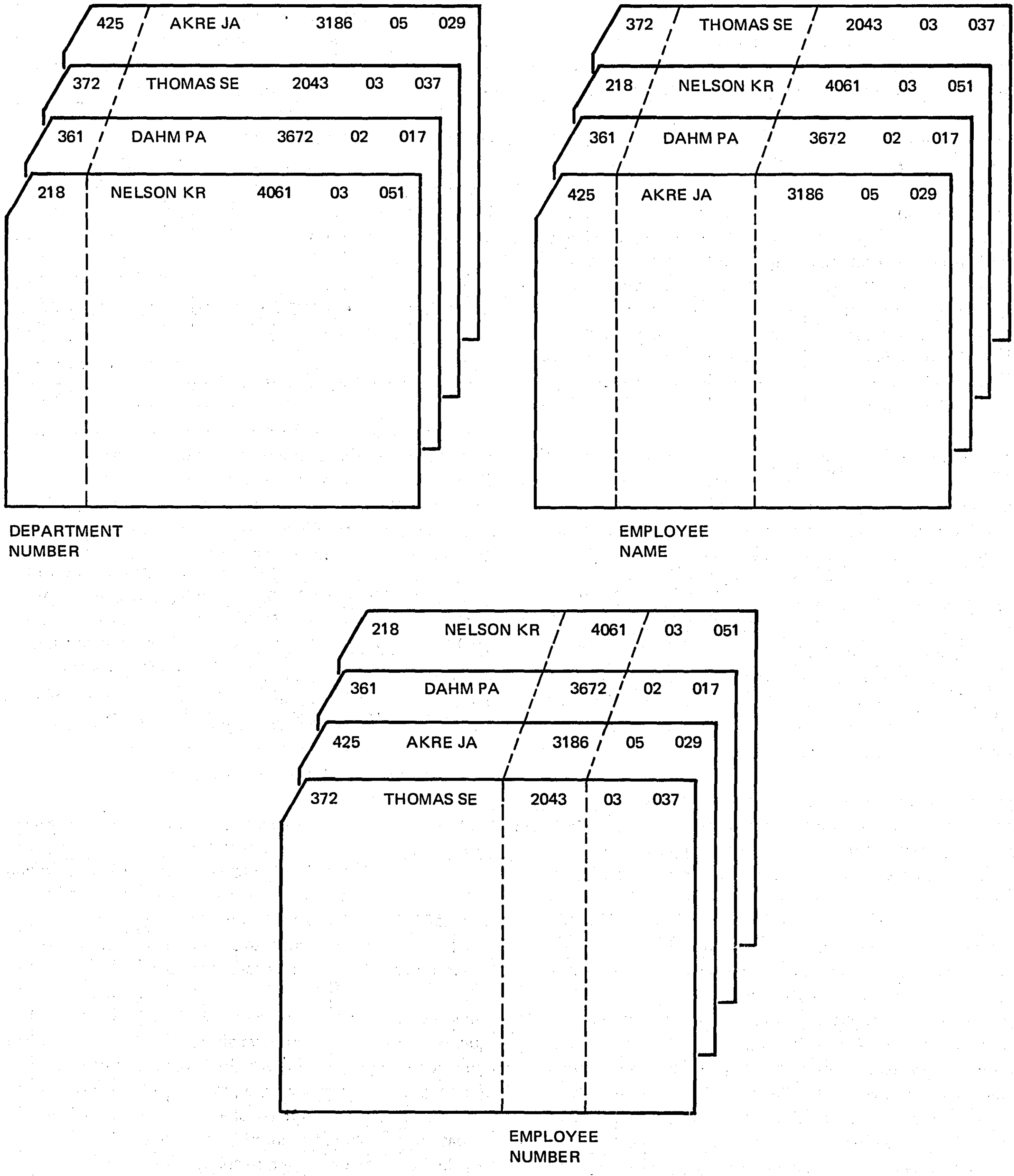
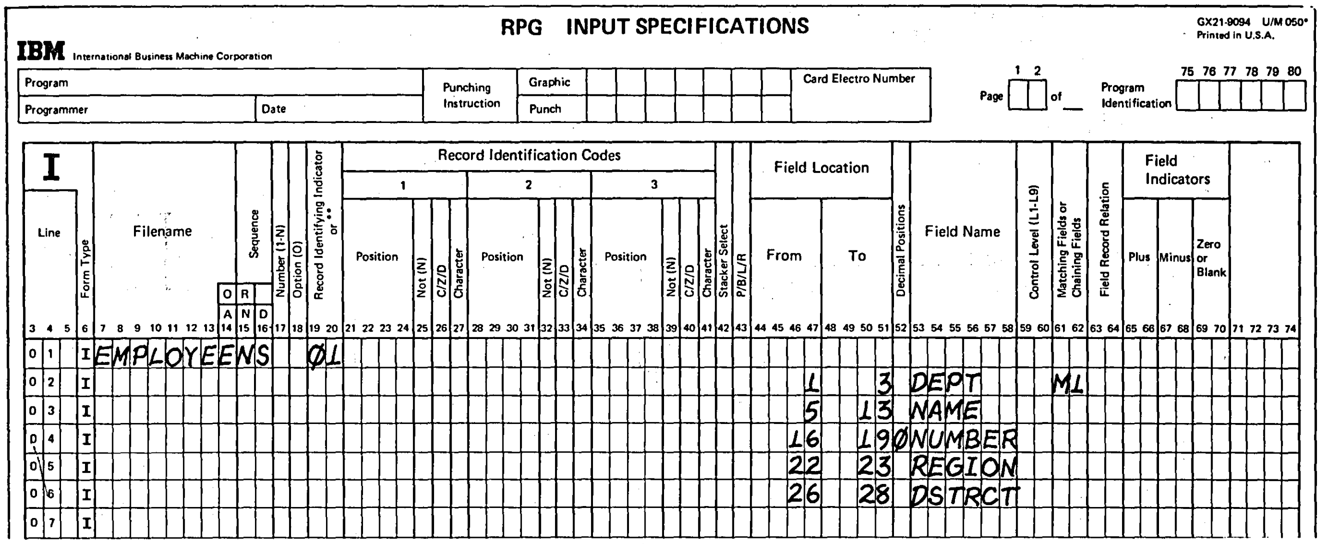


Figure 6-1. Sequenced Files



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

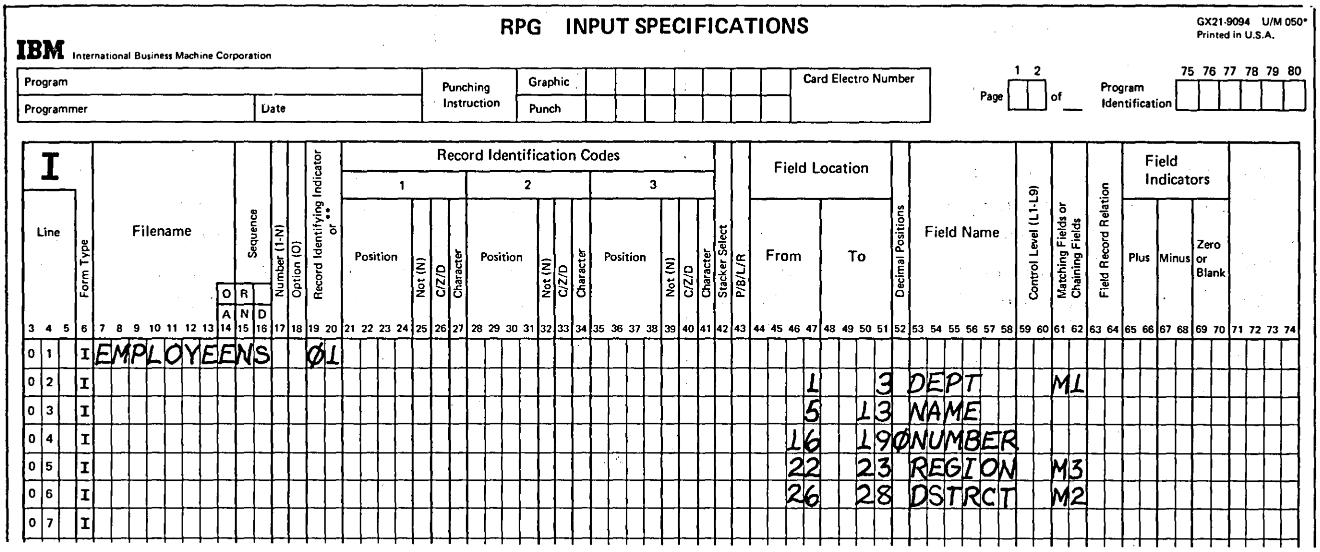
GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Record Identifying Indicator	From	To	Decimal Positions	Field Name	Matching Fields or Chaining Fields
01	I	EMPLOYEE	NS	Ø1					
02	I				1	3		DEPT	M1
03	I				5	13		NAME	
04	I				16	19	Ø	NUMBER	
05	I				22	23		REGION	
06	I				26	28		DSTRCT	
07	I								

Figure 6-2. Assigning a Match Field for Sequence Checking

The input specifications shown in Figure 6-3 show that three fields on an EMPLOYEE record are to be used for sequence checking. Since all records in a particular region are to be together, the REGION field is assigned the highest (M3) match field entry and, thus, is checked first. DSTRCT is the next match field (M2), and DEPT is the last (M1).

Figure 6-4 shows three records from the EMPLOYEE file. The three match fields shown would be considered one field. If the file is specified to be in ascending order (on the File Description sheet), the records are in order since 03037372 is lower than 03051218, which is lower than 05029425. However, if the file is to be in descending sequence, record 1 and record 3 must be switched.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Record Identifying Indicator	From	To	Decimal Positions	Field Name	Matching Fields or Chaining Fields
01	I	EMPLOYEE	NS	Ø1					
02	I				1	3		DEPT	M1
03	I				5	13		NAME	
04	I				16	19	Ø	NUMBER	
05	I				22	23		REGION	M3
06	I				26	28		DSTRCT	M2
07	I								

Figure 6-3. Assigning More than One Match Field for Sequence Checking

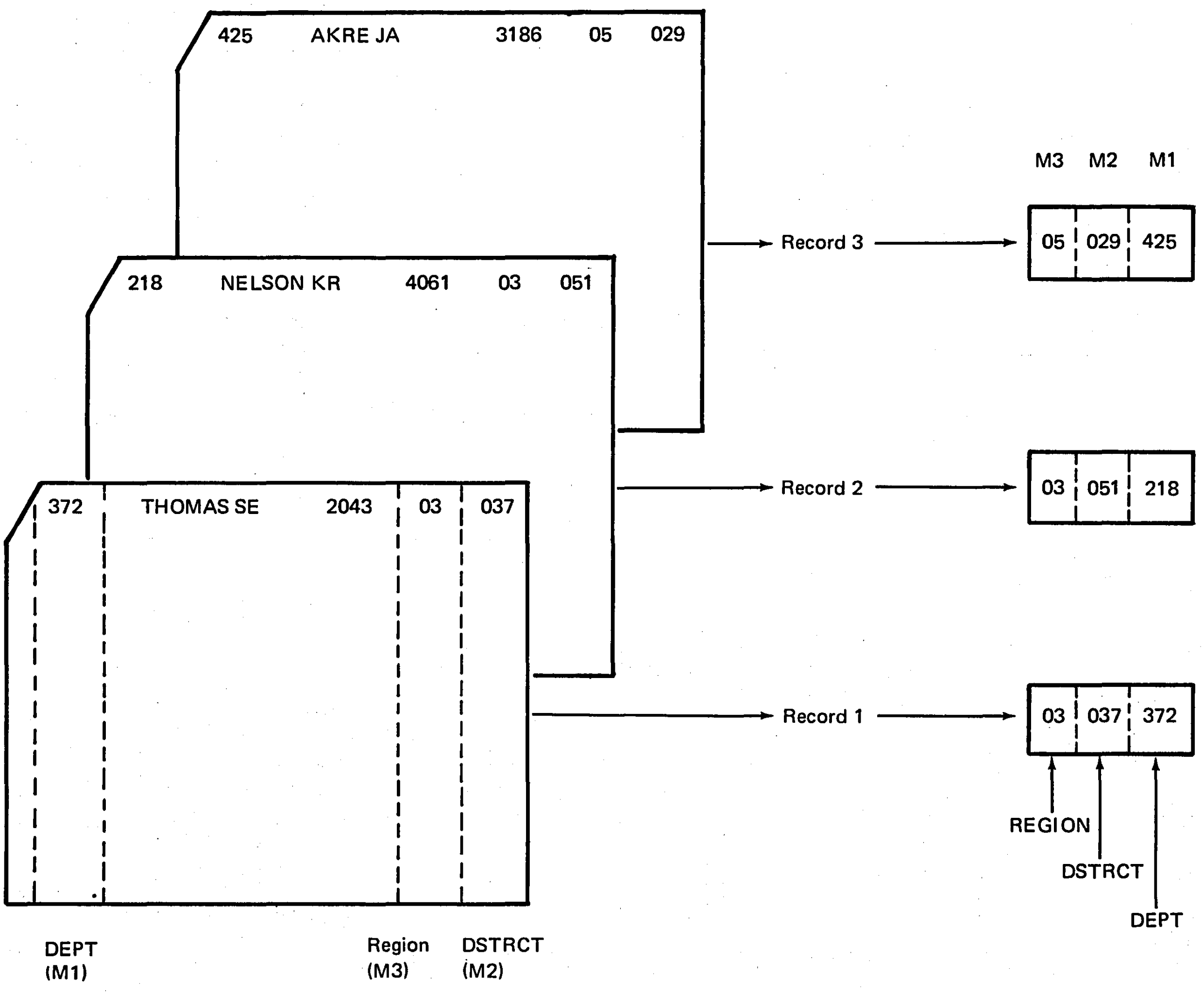


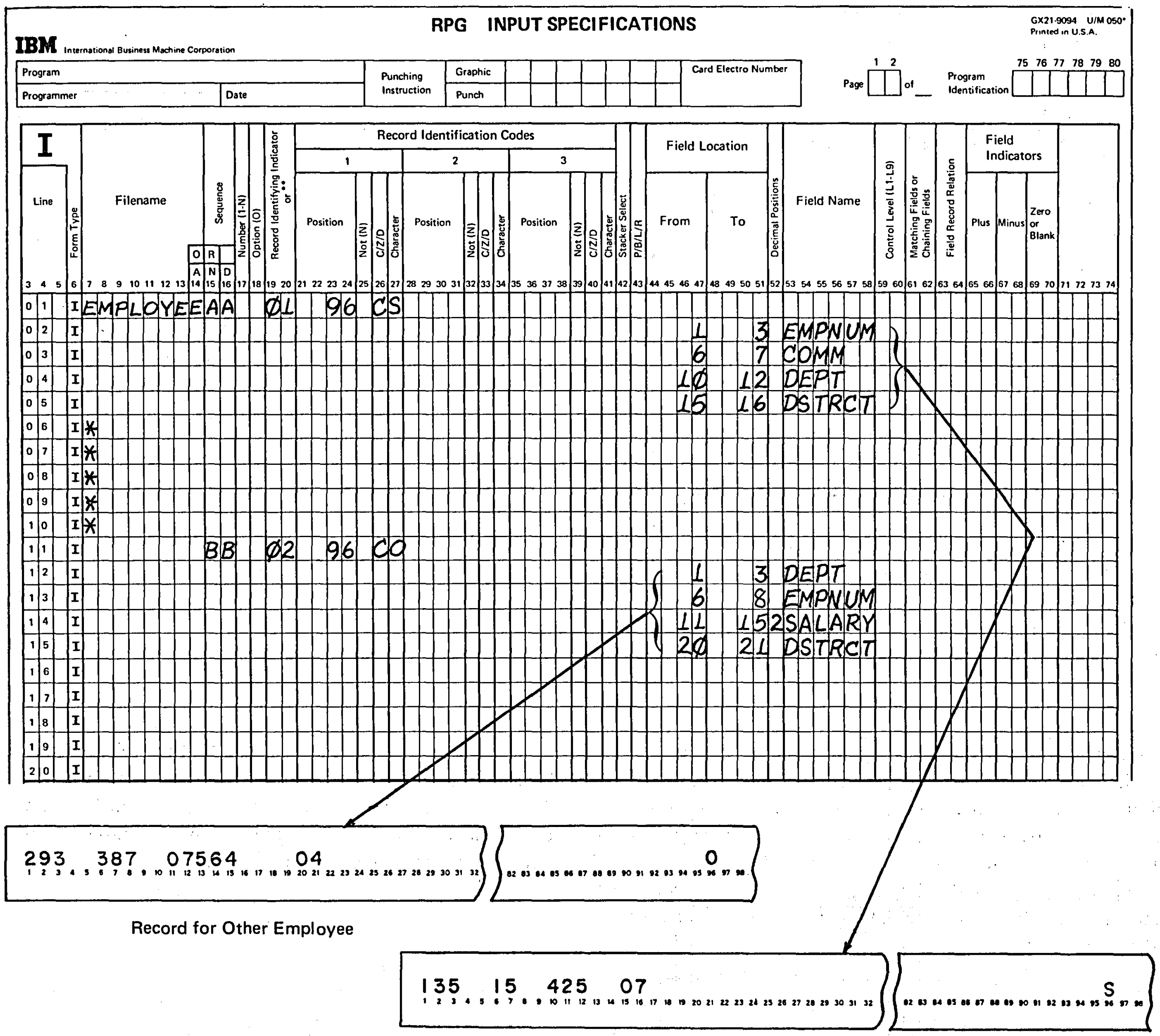
Figure 6-4. Match Fields Checked According to Level Assigned

Processing halts when the first record out of sequence is read. You can then correct the order of the records to continue processing. Note, however, that only an error in the direction of the sequence is detected. When sequence checking a file with match fields, RPG II does not cause the processing to stop when a duplicate match field is read. This can be accomplished, however, by coding the sequence check using calculation specifications.

In the last example, all records in the EMPLOYEE file were the same record type; that is, they contained the same type of information in the same location on each record. Therefore, a particular match field could always be found in certain positions for every record in the file.

File Containing More than One Record Type

When some or all of the fields on a record are in different positions than the fields on other records in the same file, you have different record types. Suppose your EMPLOYEE file is made up of two different record types, one type for salesmen and one type for all other employees. An *S* in position 96 identifies records of salesmen; an *O* identifies the other employee records. Two different record types are necessary because salesmen receive a percent commission on sales, while other employees receive a set weekly salary. Although commission and salary fields appear on only certain records, all EMPLOYEE records contain an employee number, department, and district, as shown in Figure 6-5. However, these common fields appear in different locations on different record types.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	Character	From	To	Decimal Positions	Field Name
01	I	EMPLOYEE	AA	Ø1	96	CS				
02	I						1	3		EMPNUM
03	I						6	7		COMM
04	I						1Ø	12		DEPT
05	I						15	16		DSTRCT
06	I*									
07	I*									
08	I*									
09	I*									
10	I*									
11	I		BB	Ø2	96	CO				
12	I						1	3		DEPT
13	I						6	8		EMPNUM
14	I						11	15	2	SALARY
15	I						2Ø	21		DSTRCT
16	I									
17	I									
18	I									
19	I									
20	I									

```
293  387  07564  04  ...  0
```

Record for Other Employee

```
135  15  425  07  ...  S
```

Record for Salesman

Figure 6-5. File Containing Two Record Types

For this particular program, you want a list of all employees, in ascending sequence according to district. Furthermore, the records within a district are to be in sequence according to department and employee numbers. To ensure the sequence of the file, three fields on each record must be checked, as shown in Figure 6-6. DISTRCT is the most important category and, therefore, is assigned the highest match field entry.

Since there are two different record types, a particular match field may not always be in the same record positions. For instance, DSTRCT is in positions 15-16 on salesmen records, and in positions 20-21 on records for other employees. You must tell the program where to locate the match fields *for each record type* (Figure 6-7). Once the program determines the record type by checking the code in position 96, it then looks at the appropriate match field positions for that record type.

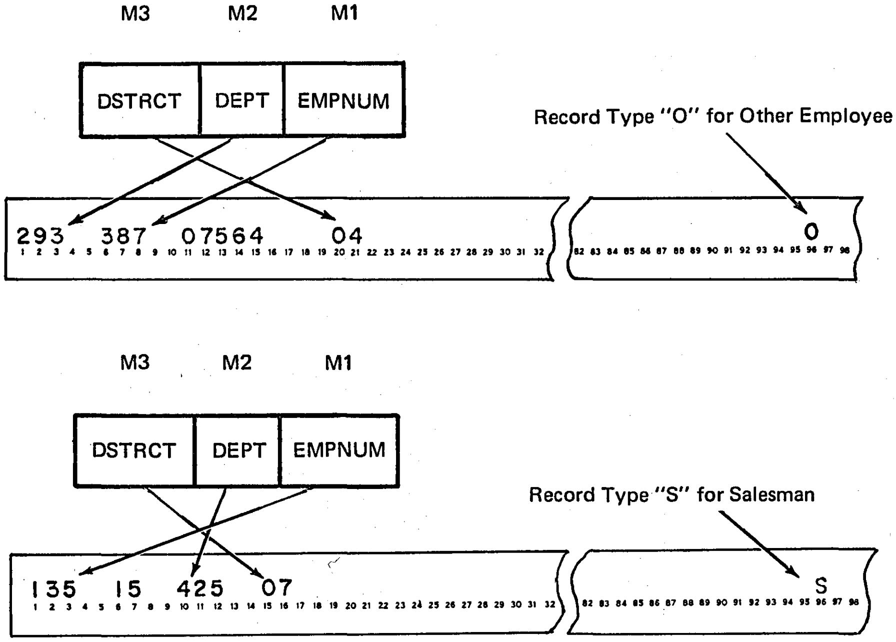
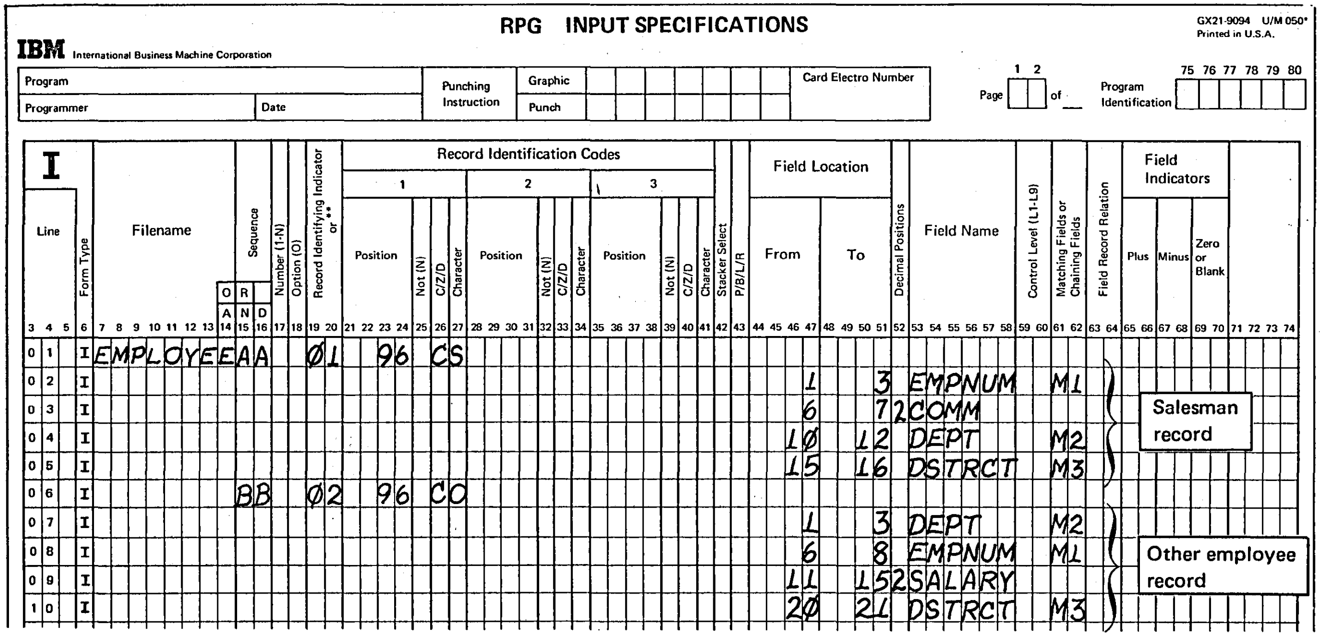


Figure 6-6. Match Fields in Different Locations on Two Records



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1 Position	Not (N)	C/Z/D	Character	2 Position	Not (N)	C/Z/D	Character	3 Position	Not (N)	C/Z/D	Character	Stacker Select	P/B/L/R	Field Location From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank
01	I	EMPLOYEE	AA			Ø1	96		C	S																				
02	I																				1	3		EMPNUM		M1				
03	I																				6	7	2	COMM						
04	I																				1Ø	12		DEPT		M2				
05	I																				15	16		DSTRCT		M3				
06	I		BB			Ø2	96		C	O																				
07	I																				1	3		DEPT		M2				
08	I																				6	8		EMPNUM		M1				
09	I																				11	15	2	SALARY						
10	I																				2Ø	21		DSTRCT		M3				

Salesman record

Other employee record

Figure 6-7. Assigning Match Fields to Different Record Types

Fields from different record types which have been assigned the same match value may have the same name. As shown in Figure 6-7, M2 has been assigned both to the DEPT field on salesman records and to the DEPT field on other employee records.

If match fields are assigned to more than one record type in a program, all of the records (with match fields) must be assigned the same number of match fields. Furthermore, all match fields (on different record types) which are given the same value (M1-M9) must be the same length. Thus, all M1 fields must be the same length, all M2 fields must be the same length, and so on. This, of course, means that the total length of the match fields must be the same for each of the records.

In this example, the EMPLOYEE file contains only two types of records and both are assigned match field entries. However, match fields need not be specified for all record types in a file. Record types which do not contain match fields are processed in the order they are read. The sequence check, then applies only to the record types to which match field entries have been assigned.

USING MATCH FIELDS WITH FIELD-RECORD RELATION FOR MORE THAN ONE RECORD TYPE IN A FILE

In the last example, each of the record types in the EMPLOYEE file is described with a separate set of input specifications. Since all of the fields to be used for matching are in different columns on the different record types, the same match field entries are assigned once for each record type.

Match Fields the Same for All Record Types

Often, however, you may find that although there are different record types in a file, many of the fields are the same on all record types. That is, many fields have the same name, contain the same type of data, and are always found in particular positions of any record type in the file. For example, salesmen records and other employee records might be organized as shown in Figure 6-8. For both record types, all fields are the same except the COMM and SALARY fields and the record identifying code in position 96.

When only a few fields differ, record types can be described on the Input sheet in an OR relationship. Instead of using separate sets of input specifications, common fields need to be described only once. As shown in Figure 6-9, entries under Field Record Relation (columns 63-64) can then identify the fields which are unique to a particular record type. Notice that fields which are the same for all record types are described first. All fields related to a particular record type are then described before specifying the fields related to one of the other record types.

DSTRCT, DEPT, and EMPNUM are the three match fields to be used in sequence checking the EMPLOYEE file.

Since they are described only once on the Input sheet without any field record relation entries, the match field entries also need be assigned only once (Figure 6-9). When record types are described in an OR relationship and a match field entry is assigned to a field without any field record relation entry, the match field will be used for all record types.

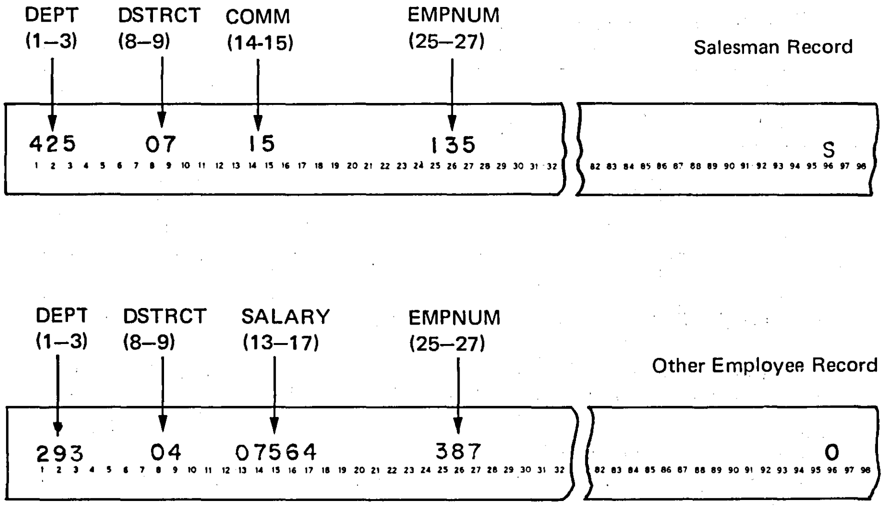
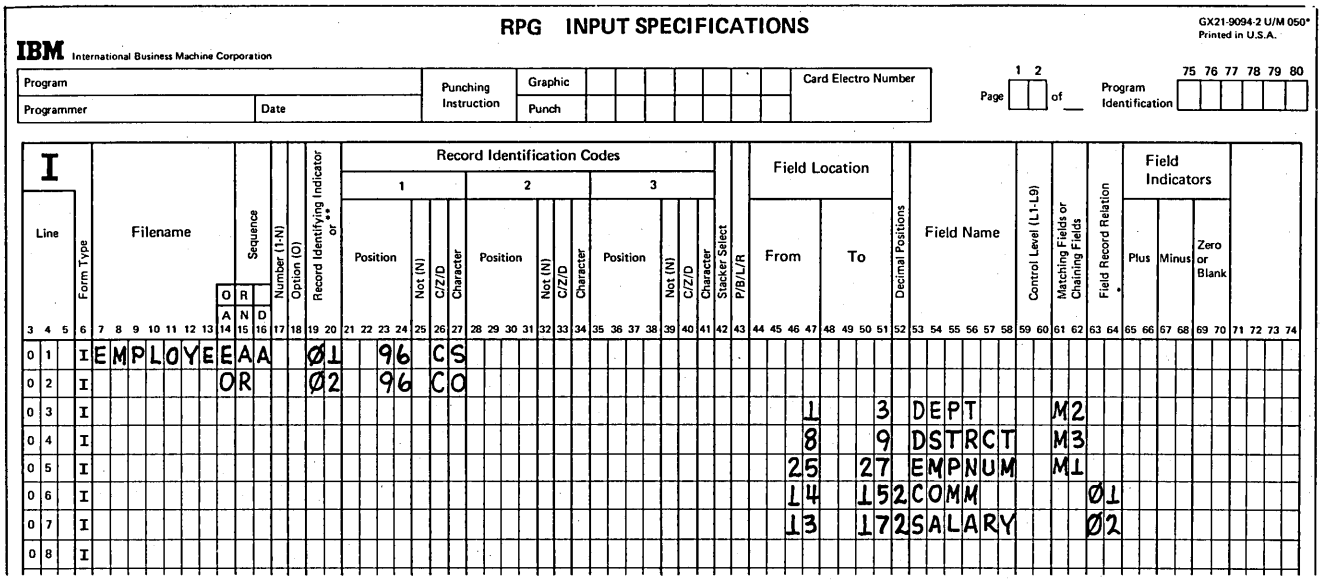


Figure 6-8. Same Match Fields for Both Record Types



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094-2 U/M 050*
Printed in U.S.A.

Program

Programmer

Date

Punching Instruction: Graphic / Punch

Card Electro Number

Page of

Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation
01	I	EMPLOYEE	AA			Ø1	96		C	S							
02	I	OR				Ø2	96		C	O							
03	I										1	3		DEPT		M2	
04	I										8	9		DSTRCT		M3	
05	I										25	27		EMPNUM		M1	
06	I										14	15	2	COMM			Ø1
07	I										13	17	2	SALARY			Ø2
08	I																

Figure 6-9. Assigning Match Fields Once for Two Record Types

Match Fields Differ Between Record Types

In the last example, although some fields differed between record types, all the fields to be used in matching were the same (same name, format, and record positions). Suppose, however, that one of the match fields, DEPT, is in different record positions for each record type. The two record types in the EMPLOYEE file are shown in Figure 6-10.

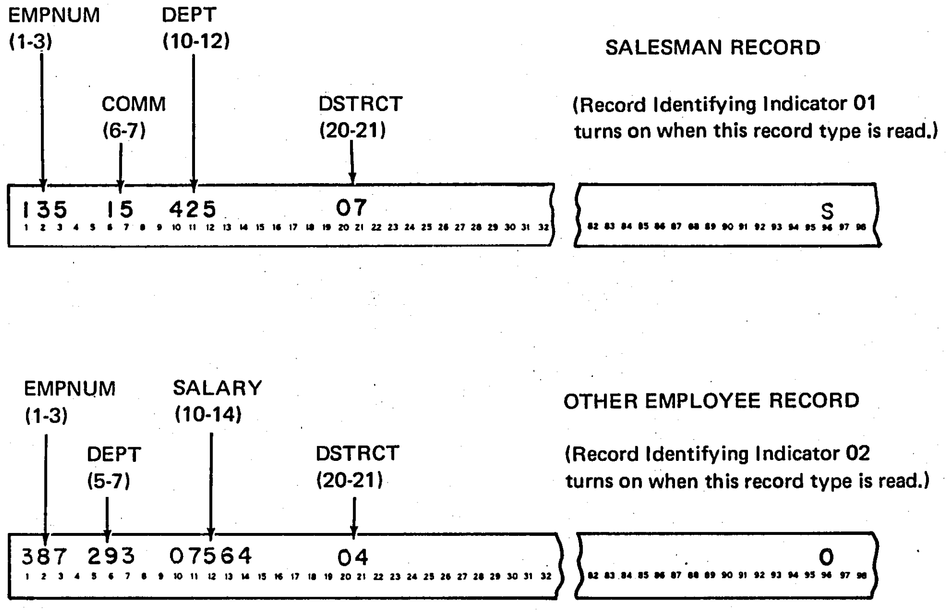


Figure 6-10. Match Fields Differ Between Record Types

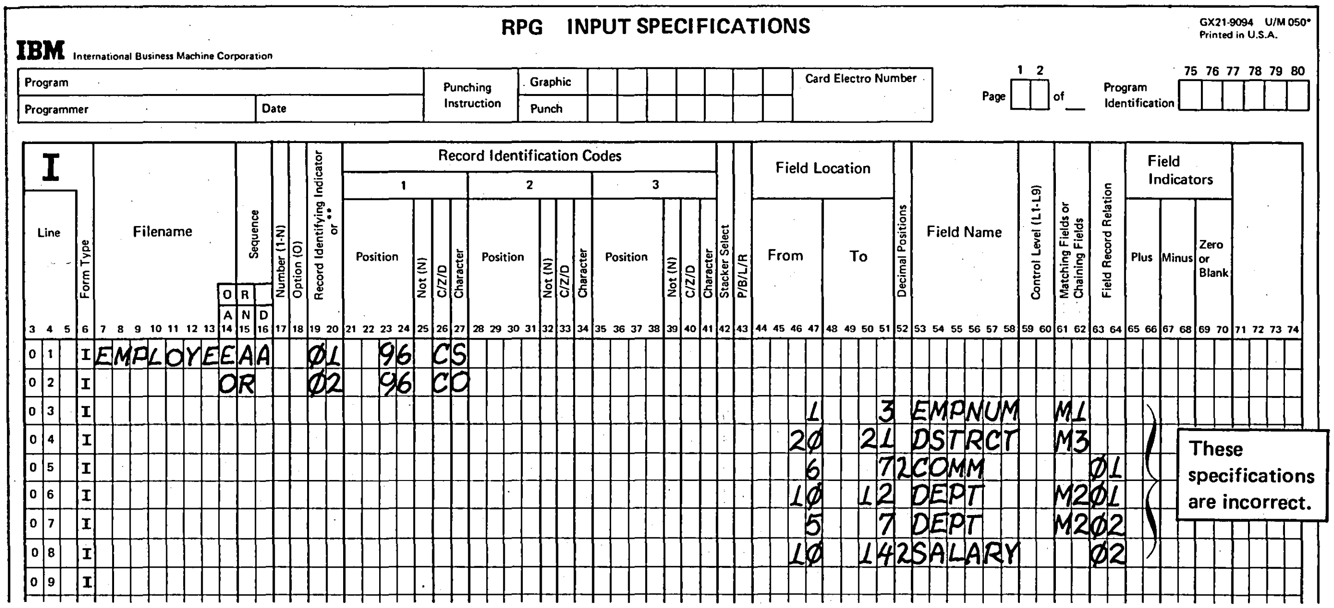
According to what you have learned so far, you might assign match field entries as shown in Figure 6-11.

However, the specifications shown in Figure 6-11 will *not* work. There is an additional rule to remember in assigning match field entries when field-record relation is used and the match fields differ between record types. If one (or more) of the match fields in either file is assigned without a field-record relation, the rest of the match fields must be assigned in the same way, without entries in columns 63-64. Of course, the specifications which assign some of these match fields with field-record relation are still necessary. Notice in Figure 6-11 that two of the match fields (M1 and M3) are associated with all record types (without field-record relation entries). Therefore, an additional entry (dummy entry) should be made (see line 05 of Figure 6-12) to assign the M2 match field (DEPT) to all record types also.

Actually the M2 match field isn't the same for all record types, since the location of the field varies. Therefore, how do you know which entries to make in the Field Location columns of this dummy match field entry? You know that any one match field is always the same length, regardless of record type or location in the record. In this case, DEPT is three positions long. Any numbers which give the correct length of the match field can be specified as Field Location.

As shown in Figure 6-12, positions 12-14 are specified for the dummy match field. When this specification line is performed, the program looks at the three positions of data (positions 12-14) on whichever record type was read. Of course, the M2 match field is not in positions 12-14 on either record type so you don't want this incorrect data to be used in sequence checking. It won't be, because *all of the match field entries are checked before the sequence checking is performed.* If the record read is record type 01, line 07 is performed. This entry tells the program that the data in positions 10-12 should be used for the M2 field. On the other hand, if record type 02 is read, line 08 is performed and the data in positions 5-7 is used instead of that in positions 12-14. Actually, then, when either of the specifications in lines 07 or 08 is performed (depending on record type), the data used as the match field is changed, as if the dummy entry had never been specified.

Although the specifications in Figure 6-12 will cause the EMPLOYEE file to be sequence checked correctly, there is a way to reduce the number of specifications required. As mentioned, for a dummy entry you can specify any record positions which give the correct length for the match field. However, if you specify the actual positions associated with that match field on one of the record types, there is no need for the specification which relates those positions to the match field for that record type (Figure 6-13). Thus, by entering 10 to 12 (line 05, Figure 6-13) as the positions for the M2 field, you can eliminate the match field entry (line 07) in which the M2 field is described for record type 01.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

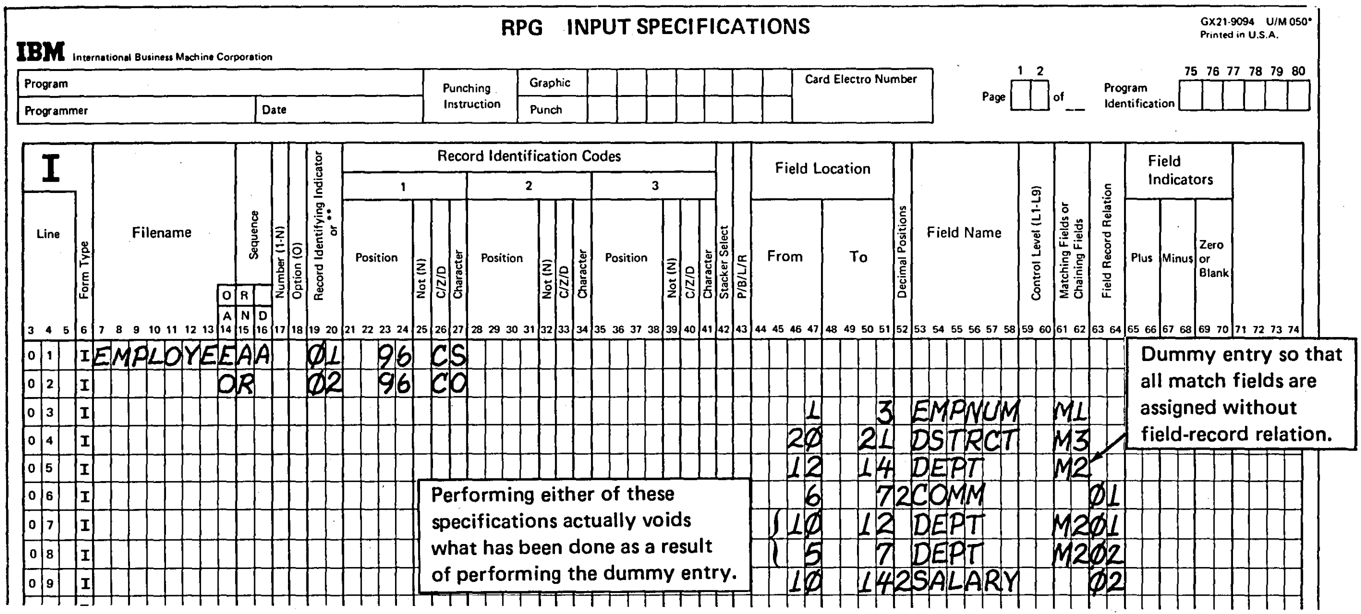
GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	C/Z/D	Character	From	To	Decimal Positions	Field Name	Matching Fields or Chaining Fields	Field Record Relation
01	I	EMPLOYEE	AA	Ø1	96	C	S						
02	I		OR	Ø2	96	C	O						
03	I							1	3		EMPNUM	M1	
04	I							2Ø	21		DSTRCT	M3	
05	I							6	7	2	COMM		Ø1
06	I							1Ø	12		DEPT	M2	Ø1
07	I							5	7		DEPT	M2	Ø2
08	I							1Ø	14	2	SALARY		Ø2
09	I												

These specifications are incorrect.

Figure 6-11. Incorrect Match Field Entries for Records Described with Field Record Relation



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

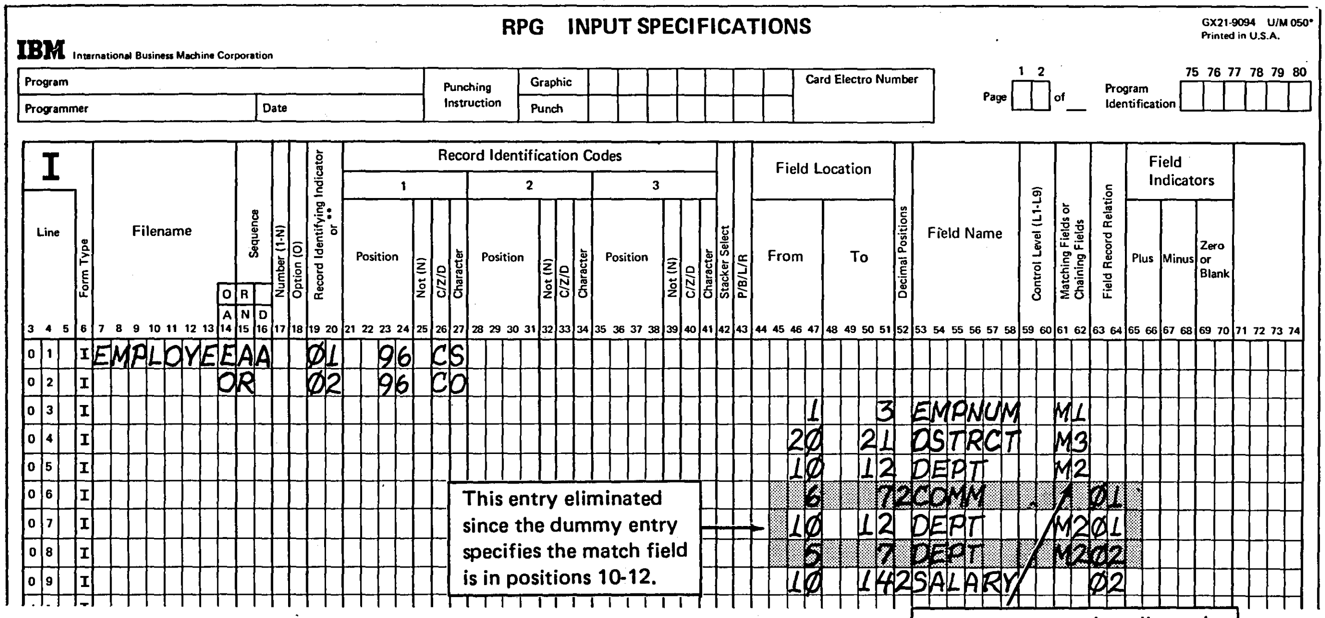
Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	C/Z/D	Character	From	To	Decimal Positions	Field Name	Matching Fields or Chaining Fields	Field Record Relation
01	I	EMPLOYEE	AA	Ø1	96	C	S						
02	I	OR		Ø2	96	C	O						
03	I							1	3		EMPNUM	M1	
04	I							2Ø	21		DSTRCT	M3	
05	I							12	14		DEPT	M2	
06	I							6	7	2	COMM		Ø1
07	I							1Ø	12		DEPT	M2	Ø1
08	I							5	7		DEPT	M2	Ø2
09	I							1Ø	14	2	SALARY		Ø2

Dummy entry so that all match fields are assigned without field-record relation.

Performing either of these specifications actually voids what has been done as a result of performing the dummy entry.

Figure 6-12. Assigning Match Fields for Records Described With Field Record Relation



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	C/Z/D	Character	From	To	Decimal Positions	Field Name	Matching Fields or Chaining Fields	Field Record Relation
01	I	EMPLOYEE	AA	Ø1	96	C	S						
02	I	OR		Ø2	96	C	O						
03	I							1	3		EMPNUM	M1	
04	I							2Ø	21		DSTRCT	M3	
05	I							1Ø	12		DEPT	M2	
06	I							6	7	2	COMM		Ø1
07	I							1Ø	12		DEPT	M2	Ø1
08	I							5	7		DEPT	M2	Ø2
09	I							1Ø	14	2	SALARY		Ø2

This entry eliminated since the dummy entry specifies the match field is in positions 10-12.

Dummy entry so that all match fields are assigned without field-record relation.

Figure 6-13. Eliminating Specifications in Assigning Match Field Entries

After performing the dummy entry (line 05), the program knows the M2 match field is to be found in positions 10-12. If record type 01 is read, the M2 field actually is in positions 10-12. Thus, line 07 doesn't have to be performed, since it does not change anything. Of course, if record type 02 is read, the specification in line 08 is performed. This says, for record type 02, use positions 5-7 for the M2 field instead of positions 10-12.

The field name specified for the dummy entry can be any name, since field names are ignored in selecting match fields. In this case, DEPT was specified since it happens that the M2 fields on both record types have the same name. If the names to be used differ, it is still a good practice to use a name given to the match field on one of the record types.

MATCHING RECORDS: ONE RECORD TYPE IN EACH FILE

One of the most common forms of multifile processing involves using one file to obtain data from another file. Figure 6-14 shows a weekly sales report to be printed which is used to determine which items are selling best at which location. A SALES file contains records of individual items sold, giving the quantity and location. The description of each item, however, must be obtained from an ITEM master file. The ITEM master file consists of one ITEM record for each item in stock.

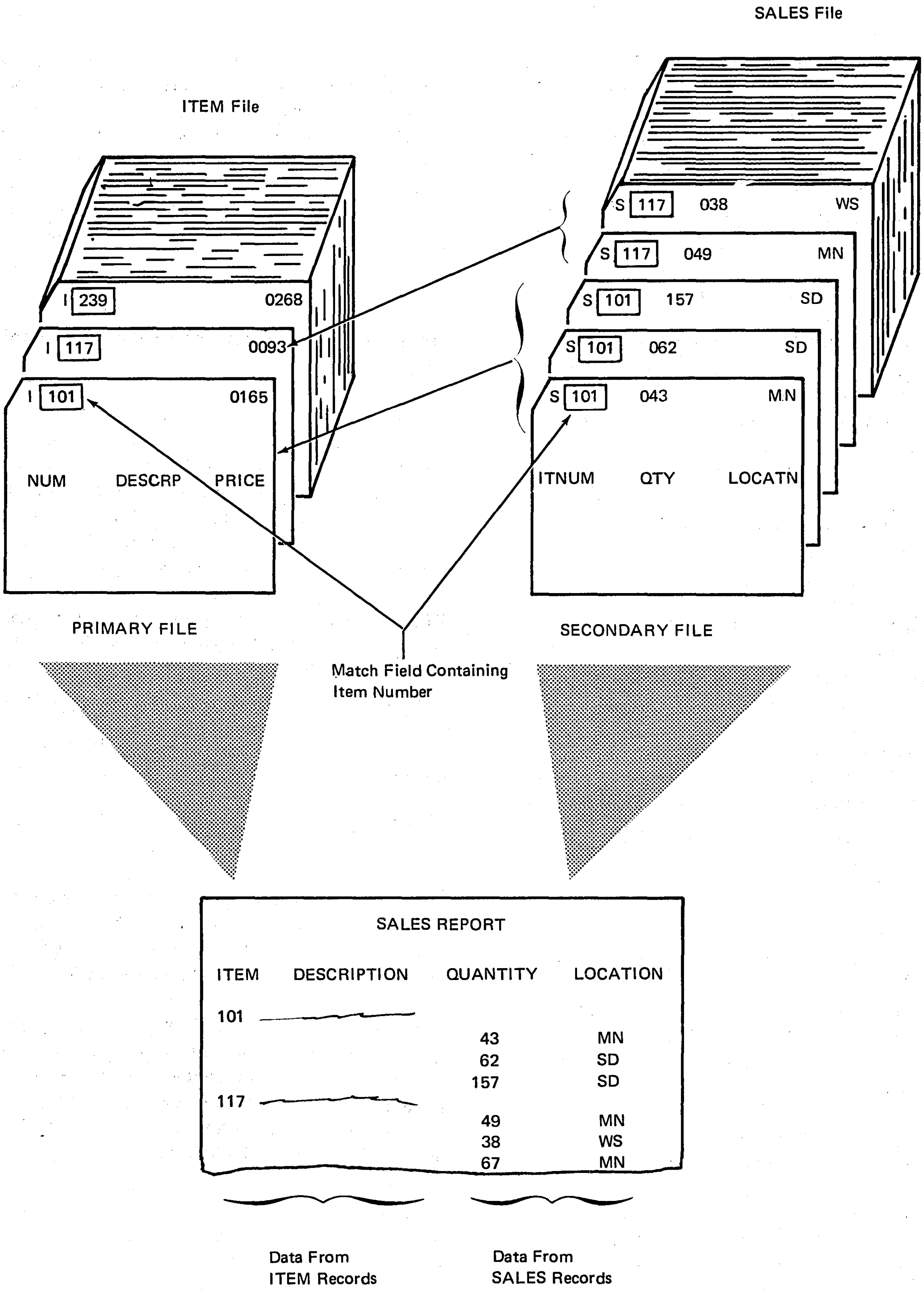
For this program, let's assume that each file contains only one record type. All ITEM records are in one format and are identified by the character I in position 1. All records in the SALES file are identified by an S in position 1 and are also in one format (certain type of information in same location on every record). The SALES records for a particular item can be associated with the related ITEM master record by a common match field containing the item number (see Figure 6-14).

Processing Order: More Than One Matching Record in a Secondary File

In the ITEM master file, there is only one record for each item, but in a program run there may be several SALES records for that particular item. Let us suppose there is always at least one SALES record for each ITEM record. Therefore, there should be no records in either file which do not have a matching record in the other file. Both files are specified with match fields in ascending sequence.

To produce the report in Figure 6-14, in what order should records from the two files be processed? First an ITEM master record should be processed; that is, the item number and description printed. Then, all the SALES records which are for the same item (match fields the same) should be processed. The quantity and location of each sale should be printed under description. Thus, after every ITEM record processed, one or more SALES records must be processed before the next record from the ITEM master file is processed.

How does RPG II know when to stop printing SALES records and to process the next ITEM? As we said, the SALES records for a particular item are read one at a time and printed. When the match field on a SALES record is for a different item, that SALES record is not processed immediately. Instead, the next ITEM master record is processed to print the description for the new item. Then the SALES records for that item are printed. As you can see, to determine the correct order of processing, both files must be in the same sequence according to the match field; in this case, in ascending sequence by item number.



SALES REPORT

ITEM	DESCRIPTION	QUANTITY	LOCATION
101			
		43	MN
		62	SD
		157	SD
117			
		49	MN
		38	WS
		67	MN

Figure 6-14. Matching SALES Records with Related ITEM Records to Produce a Printed Report

How RPG II Logic Determines From Which File to Process a Record

Using the same example, let's see how the RPG II logic determines from which file a record is to be processed. The ITEM master records are the primary file and the SALES records are the secondary file. When two input files are used in a program, the program cycle is slightly different for the first record read. You can follow the logic flow shown in Figure 6-15 as we discuss the first program cycle and subsequent cycles for this job. Pay special attention to when the record identifying indicator is on, identifying the record to be processed, and when the MR indicator is turned on and off.

At the beginning of the first program cycle (Figure 6-15, part A), a record is read from each file. The first step is to determine which record to process. The program determines if match fields are specified for both record types. In this case, both the ITEM record and the SALES record contain an M1 field. The match fields from each file are then compared to see which is lower in sequence. (If the file had been in descending sequence, the program would check for the highest match field.) In this case, neither field is lower as the match fields on the primary and secondary records are both 101. When match fields from the two files are the same, the record from the primary file is always selected for processing.

Now the appropriate record identifying indicator is turned on to identify the type of record selected for processing. Thus, 01 would be turned on for the ITEM record from the primary file.

Once a record is selected for processing and the record identifying indicator is turned on, the program determines whether the MR indicator should be on during processing of the record. Since the match fields are equal (both 101), a matching record condition exists. The MR indicator is not turned on yet, however, because the selected record is not ready to be processed yet.

First, the program checks to see if any total operations are to be performed for previously processed records. Total calculations and total output are performed only if control fields change or if the last record in the file has already been processed (LR on). Since there are no control fields in either file, control level indicators will never be turned on during this program. Furthermore, this is not the last record. Neither condition has occurred; therefore, total-time operations are bypassed.

At this point, the MR indicator is turned on to indicate that the matching record conditon exists. Now the program is ready to perform the detail operations for the record which was selected for processing. Thus, the item number and description from the primary file record are printed. The program knows which operations are to be performed because the operations are either conditioned to be performed only when record identifying indicator 01 and the MR indicator are on or they are not conditioned by indicators.

Once the processing of this record is completed, the record identifying indicator is turned off (01 off). This completes the first program cycle; that is, one record has been processed.

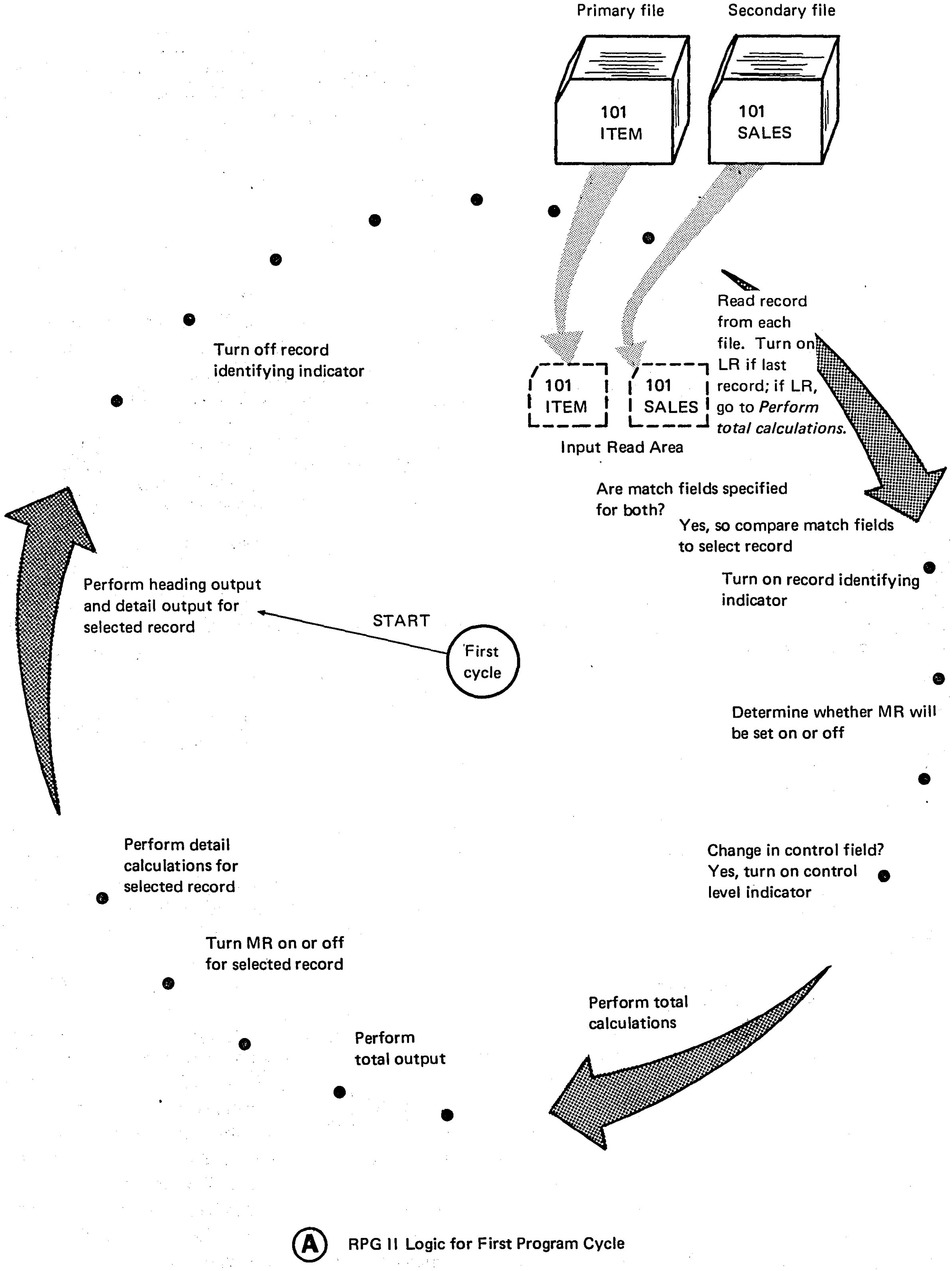
For the processing of all subsequent records, look at the program cycle in Figure 6-15, part B. The entire cycle will be repeated for each record processed.

At the beginning of the first cycle, a record was read from both the primary file and the secondary file. However, since only one record is selected for processing at a time, one record (in this case, a SALES record) still remains in the input read area. Therefore, a record from only one file is read at the beginning of the second cycle and for all following program cycles. The record read will be from the same file as the last record processed. Thus, for the second cycle, the second record from the ITEM file would be read into the primary file read area.

Now that a record from each file is in the read area again, the second record can be selected for processing. Once again the first step in the logic is to determine if match fields are specified for the records from both files. In this program, there is only one record type per file and both are assigned match fields. Therefore, the answer to this question is yes for every program cycle of this program.

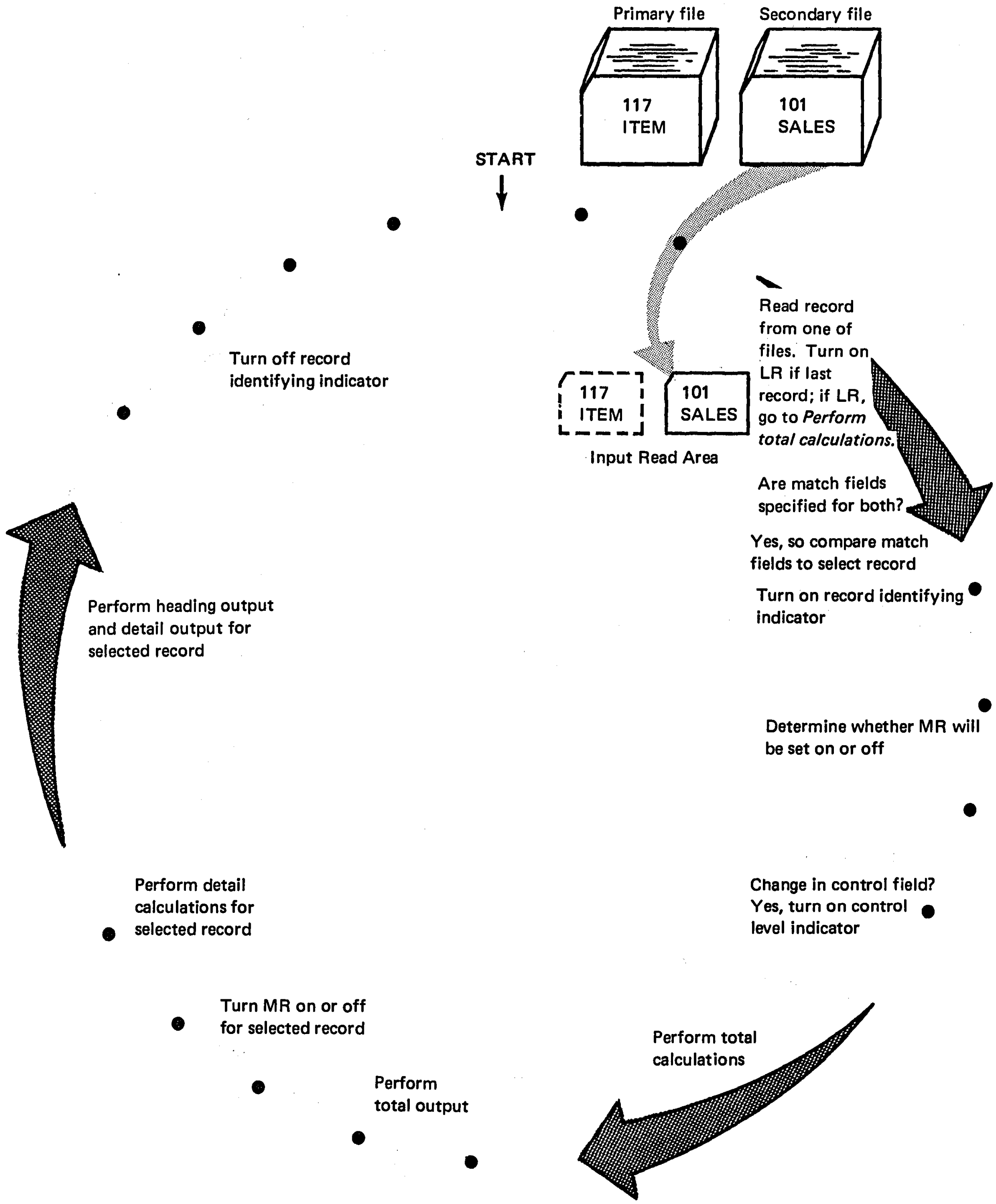
The match fields from both records are then compared. In this case, the secondary file SALES record is for the first item number (101). Since the ITEM record for the first item (101) has already been processed, the primary file record in the read area is for the second item number (117). The match field on the SALES record is lower in sequence than the match field on the ITEM record; therefore, the SALES record is selected to be processed. Once again, a record identifying indicator (02 for a SALES record) is turned on to identify the type of record selected.

At this point, the MR indicator is still *on* because the previous comparison (item 101) did give a match. The setting of MR has nothing to do with the record which was just selected for processing. The indicator will not be set to reflect the current condition until just before the record selected for processing (SALES record for item 101) is processed.


Primary file
Secondary file
101
ITEM
101
SALES
Read record from each file. Turn on LR if last record; if LR, go to *Perform total calculations.*
101
ITEM
101
SALES
Input Read Area
Are match fields specified for both?
Yes, so compare match fields to select record
Turn on record identifying indicator
Determine whether MR will be set on or off
Change in control field? Yes, turn on control level indicator
Perform total calculations
Perform total output
Turn MR on or off for selected record
Perform detail calculations for selected record
Perform heading output and detail output for selected record
START
First cycle
Turn off record identifying indicator
A

RPG II Logic for First Program Cycle

Figure 6-15 (Part 1 of 2). Logic of Matching Records


Primary file
Secondary file
117
ITEM
101
SALES
START
Read record from one of files. Turn on LR if last record; if LR, go to *Perform total calculations.*
117
ITEM
101
SALES
Input Read Area
Are match fields specified for both?
Yes, so compare match fields to select record
Turn on record identifying indicator
Determine whether MR will be set on or off
Change in control field? Yes, turn on control level indicator
Perform total calculations
Perform total output
Turn MR on or off for selected record
Perform detail calculations for selected record
Perform heading output and detail output for selected record
Turn off record identifying indicator

(B) RPG II Logic for Subsequent Program Cycles

Figure 6-15 (Part 2 of 2). Logic of Matching Records

Since the next record to be processed has been selected, the RPG II program now determines how the MR indicator is to be set for processing this card. Is there a matching record condition? You are probably thinking that there is not, because the match fields are not the same (secondary file, 101; primary file, 117). However, when the match fields are not equal and a *secondary file* record has been selected, the match field on the secondary file record (SALES) is then compared with the match field of the last primary file record processed. The last ITEM record processed has a match field of 101, the same as the match field of the SALES record to be processed. Therefore, there is another matching record condition.

The MR indicator is not set again and the record selected (SALES 101) is not processed (detail operations not performed) until after any total operations to be done for previous records are completed. Since there are no control fields and this is not the last record in the file, total operations can be skipped in this program cycle.

At this point, immediately before processing the selected record, the MR indicator is set. Although MR is already *on* for the previous record processed, it is set *on* again for the processing of this record (SALES record for item 101). All detail operations conditioned for record type 02 (or for both 02 and MR indicators on) are then performed. Thus, the quantity and location fields on the SALES record are printed.

After printing the SALES record, processing for this record is complete and record identifying indicator 02 is turned off. In the next program cycle, another SALES record is read from the secondary file to replace the SALES record just processed.

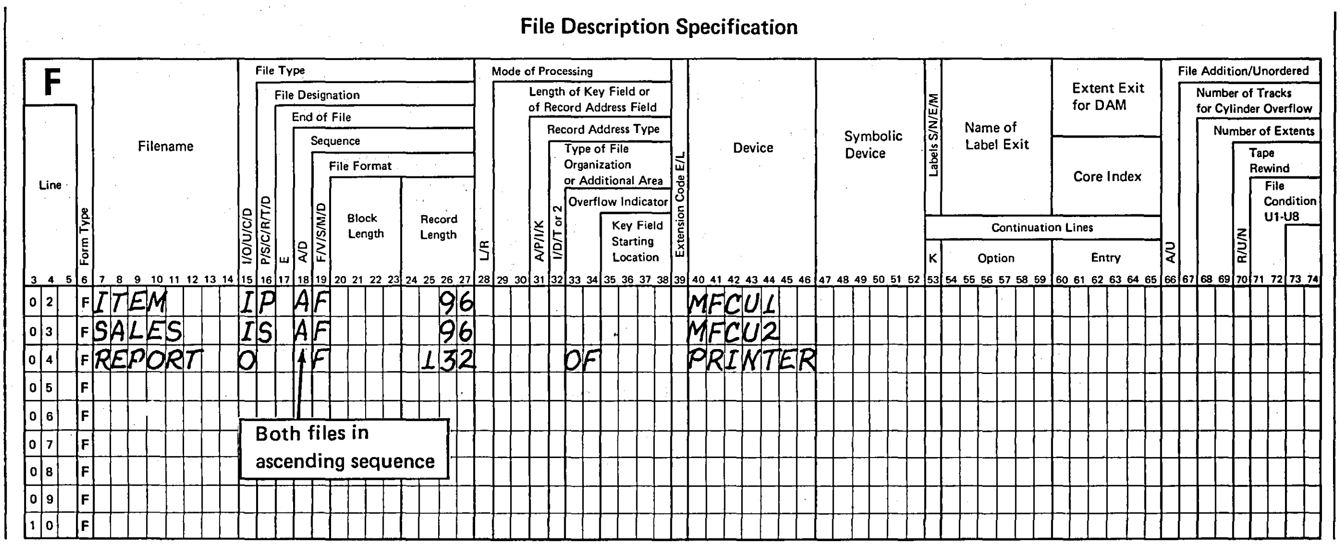
For the rest of the SALES records related to the first item number (101), the comparison of match fields would indicate that the secondary file records are lower in sequence than the item number (117) on the ITEM record of the primary file. Thus, all sales records for item 101 are processed. Furthermore, the MR indicator will be on during processing of all the records, since the match fields (101) will always be the same as the match field on the last primary record processed.

At the beginning of the next program cycle, a SALES record for the next item number (117) would be read. On comparing the SALES record match field with the match field on the ITEM record in the read area, there is a match again. The ITEM for item 117 would then be processed before the SALES records for that same item.

Specifications for a Matching Records Program

Figure 6-16 shows the three specification sheets — File Description, Input, and Output — to be used for this program. Since the data from the two input files is only to be printed, calculation specifications are not necessary.

The File Description sheet defines the two input files to be used in matching, as well as the printer file, on which the report is to be printed. Notice that the two input files must both be specified as being in the same sequence (A in column 18). The P and S entries in column 16 indicate to the system whether an input type file is to be considered as a primary or secondary file.

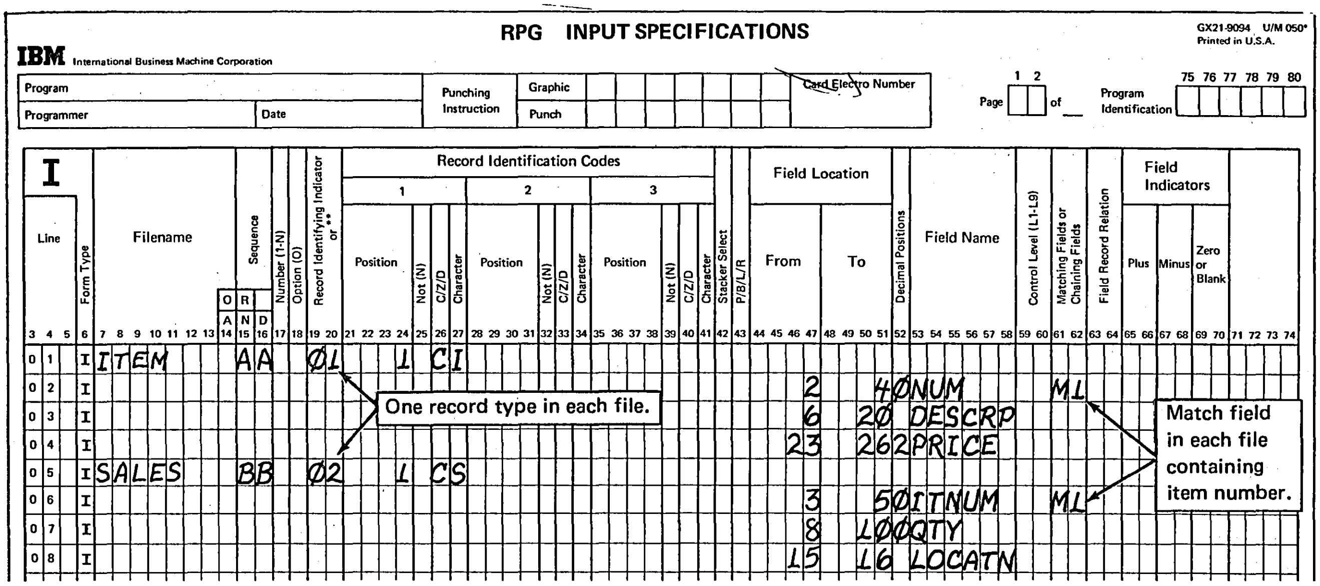


File Description Specification

Line (3-5)	Form Type (6)	Filename (7-14)	File Type I/O/U/C/D (15)	File Designation P/S/C/R/T/D (16)	End of File E (17)	Sequence A/D (18)	File Format F/V/S/M/D (19)	Block Length (20-23)	Record Length (24-27)	Mode of Processing L/R (28)	Length of Key Field or of Record Address Field (29-30)	Record Address Type A/P/I/K (31)	Type of File Organization or Additional Area I/D/T or 2 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels S/N/E/M (53)	Name of Label Exit (54-59)	Extent Exit for DAM (60-65)	A/U (66)	File Addition/Unordered (67-74)
02	F	ITEM	I	P		A	F		96								MFCU1						
03	F	SALES	I	S		A	F		96								MFCU2						
04	F	REPORT	O				F		132					OF			PRINTER						
05	F																						
06	F																						
07	F																						
08	F																						
09	F																						
10	F																						

Both files in ascending sequence

Figure 6-16 (Part 1 of 2). Specifications for Matching Records Program

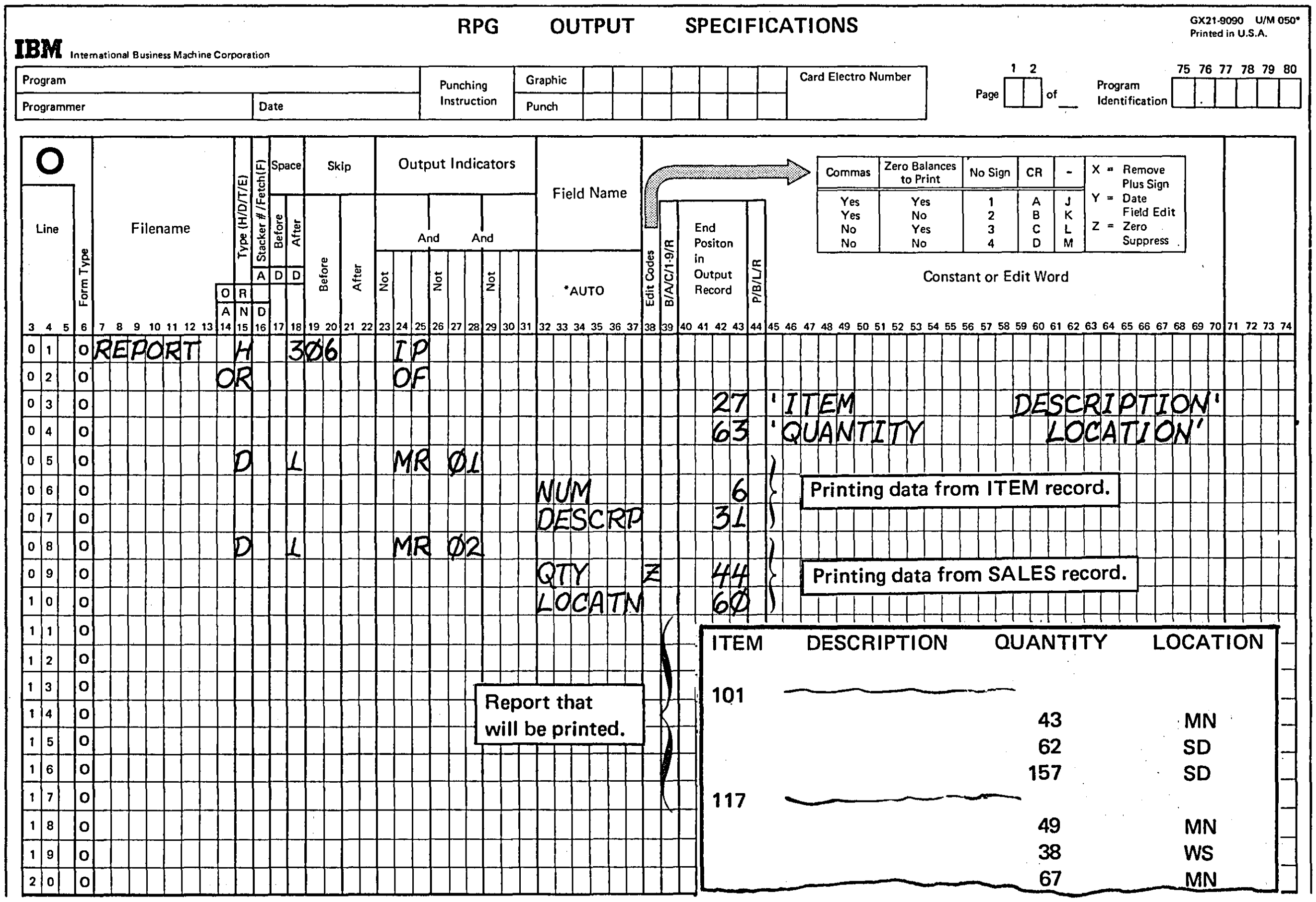


RPG INPUT SPECIFICATIONS

```
01 IITEM    AA  Ø1   1 CI
02 I                                        2   4ØNUM     M1
03 I                                        6  2Ø DESCRP
04 I                                       23  262PRICE
05 ISALES   BB  Ø2   1 CS
06 I                                        3   5ØITNUM   M1
07 I                                        8  1ØØQTY
08 I                                       15  16 LOCATN
```

One record type in each file.

Match field in each file containing item number.



RPG OUTPUT SPECIFICATIONS

```
01 OREPORT  H  3Ø6   IP
02 O       OR        OF
03 O                                   27 'ITEM            DESCRIPTION'
04 O                                   63 'QUANTITY         LOCATION'
05 O        D  1     MR Ø1
06 O                         NUM        6
07 O                         DESCRP    31
08 O        D  1     MR Ø2
09 O                         QTY   Z   44
10 O                         LOCATN    6Ø
```

Printing data from ITEM record.

Printing data from SALES record.

Report that will be printed.

```
ITEM     DESCRIPTION        QUANTITY      LOCATION

101
                               43            MN
                               62            SD
                              157            SD
117
                               49            MN
                               38            WS
                               67            MN
```

Figure 6-16 (Part 2 of 2). Specifications for Matching Records Program

File description specifications also associate each input file with a particular device. On the MFCU, the primary file is usually entered through the primary hopper (device entry MFCU1); the secondary file is usually read from the secondary hopper (device entry MFCU2). However, in multifile processing using the MFCU, either file may be associated with either of the two hoppers.

The contents of the two input files are described on the Input sheet, with a separate set of specifications for each file. Entries in columns 19-27 identify the two record types. Record identifying indicator 01 will turn on if an ITEM record is selected for processing; indicator 02 will turn on if a SALES record is selected. The single match field in each file is specified by assigning the M1 entry in columns 61-62, as you learned previously.

Lines 01-04 of the Output sheet specify that headings are to be printed at the top of every page of the report.

As you recall in the discussion of matching record logic, the 01 record identifying indicator is on whenever an ITEM record is being processed. Furthermore, ITEM records are processed only if a matching record condition exists (MR on). Therefore, conditioning the printing of the item number and description by MR and 01 ensures that the information is coming from the ITEM record.

The quantity and location are to be printed only when a SALES record is being processed. Because the MR indicator is on during the processing of both ITEM records and SALES records, it isn't sufficient to condition this output line only on the basis of MR. Therefore, the printing of quantity and location is conditioned by both the MR indicator and the 02 record identifying indicator.

In this case, calculation specifications are not required. However, if calculations are required and are to be performed only if a matching (or not matching) record condition exists, calculations must also be conditioned by the MR (or NMR) indicator.

Processing Order: More Than One Matching Record in the Primary File

For the example just presented, there was more than one record in the secondary file with the same match field. That is, a single ITEM record was related to a number of SALES records. In such a case, the order of processing was the first primary file record, followed by its related secondary file records, then the next primary file record, followed by its related secondary file records, and so on.

Let's take a look at another set of files, in which each record has a matching record in the other file. Figure 6-17, part A, shows that there may be more than one primary file record which matches the same secondary records. In what order do you think these records would be processed? The first primary file record processed would not necessarily be followed by its related secondary file records. Instead, all primary file records with the same match field AA would be processed; then all secondary file records for AA (see Figure 6-17, part B).

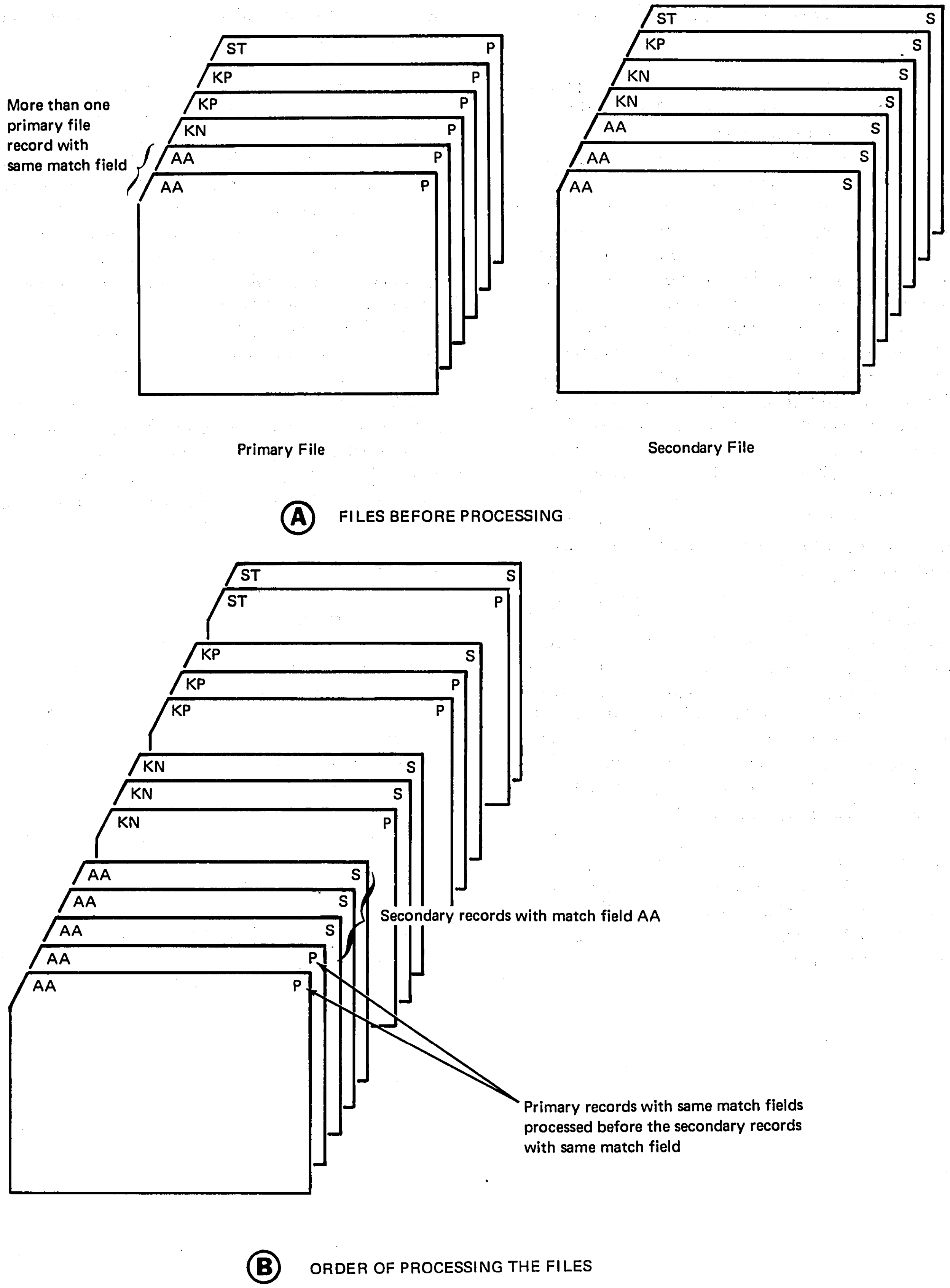
Actually, the RPG II logic used to determine the order in which such files are processed is the same as that discussed previously. Remember that whenever the match fields on the two records in the read area are the same, the primary file record is always selected for processing. Thus, during the first program cycle, the first primary file record (AA) is compared to the first secondary file record (AA). The match fields are the same, so the first primary file record is processed. Then the second primary file record (again AA) is read and compared to the first secondary file record (AA), still in the read area. Since there is a match again, this second primary file record is processed.

For both of the last two examples presented, every record in the secondary file had at least one or more related records in the primary file. Since a matching record condition always exists, the MR indicator was on during the processing of every record in both files.

Matching Records: Records Which Have No Match in the Other File

In the multifile programs presented, there was at least one SALES record for every ITEM master record. Therefore, all records had a related record in the other file. Of course, in multifile processing, it is possible to have primary file records with no related records in the secondary file, as well as secondary file records with no related primary file records. In either case, the record with no related record in the other file is called an *unmatched record.* For instance, if only certain items are sold, there would be an ITEM master record for every item and SALES records for only some of those items. The ITEM records for which no sales were made would be unmatched records.

Unmatched records can be in either or both files, but they are usually found in the primary file. In this example, it is quite probable that for any one item, no sales have taken place. However, it is not as likely that sales have occurred for an item for which there is no record in the ITEM master file.


More than one primary file record with same match field
ST P
KP P
KP P
KN P
AA P
AA P
Primary File
ST S
KP S
KN S
KN S
AA S
AA S
AA S
Secondary File
ST S
ST P
KP S
KP P
KP P
KN S
KN S
KN P
AA S
AA S
AA S
AA P
AA P
Secondary records with match field AA
Primary records with same match fields processed before the secondary records with same match field

(A) FILES BEFORE PROCESSING

(B) ORDER OF PROCESSING THE FILES

Figure 6-17. More than One Primary File Record with Same Match Field

Processing Order: Unmatched Records

Let's assume that the two files for this program consisted of the records shown in Figure 6-18, part A, with the primary file containing two unmatched records. The RPG II logic related to matching records would determine the order in which to process the records (Figure 6-18, part B). As explained earlier, first the ITEM master record for item 101 is processed, followed by all SALES records for item 101. Remember, if a record is selected for processing during a program cycle, the next record from that same file is read at the beginning of the next cycle. Therefore, when the first SALES record for item 117 is read, another matching record condition exists. Thus, the primary file record (ITEM record for item 117) is processed first. The next primary file record for item 124 is then read to replace the ITEM record processed. Since SALES records with match fields of 117 are lower in sequence than 124, the two SALES records for item 117 are processed. Furthermore, although the match fields (117) on the SALES records are not the same as that of the ITEM record in the read area (124), they are the same as the match field of the last ITEM record processed. Therefore, the MR indicator is on when each SALES record for 117 is processed.

When the next SALES record (item 239) is read, there still isn't a match with the ITEM record for 124. The match field on the ITEM record is lower than that on the SALES record; therefore, the primary file record for item 124, an unmatched record, is processed first (Figure 6-18, part B). Since the files are in ascending sequence, this means that there is no record in the secondary file with a match field of 124. Therefore, MR will be turned off just before the ITEM record for 124 is processed.

This program cycle shows that even if there is no matching record condition, the primary file may be selected for processing. This is because RPG II logic looks for the record with the lowest match field, regardless of whether the record is from the primary file or the secondary file.

Now the next primary file ITEM record (239) is read. Since it matches the SALES record in the read area, the primary file record is automatically selected for processing. Although a matching record condition exists for the ITEM record selected, at this point the MR indicator is still off because of the last record processed (unmatched ITEM record for 124). The status of MR is not changed until right before detail operations for the selected record are performed.

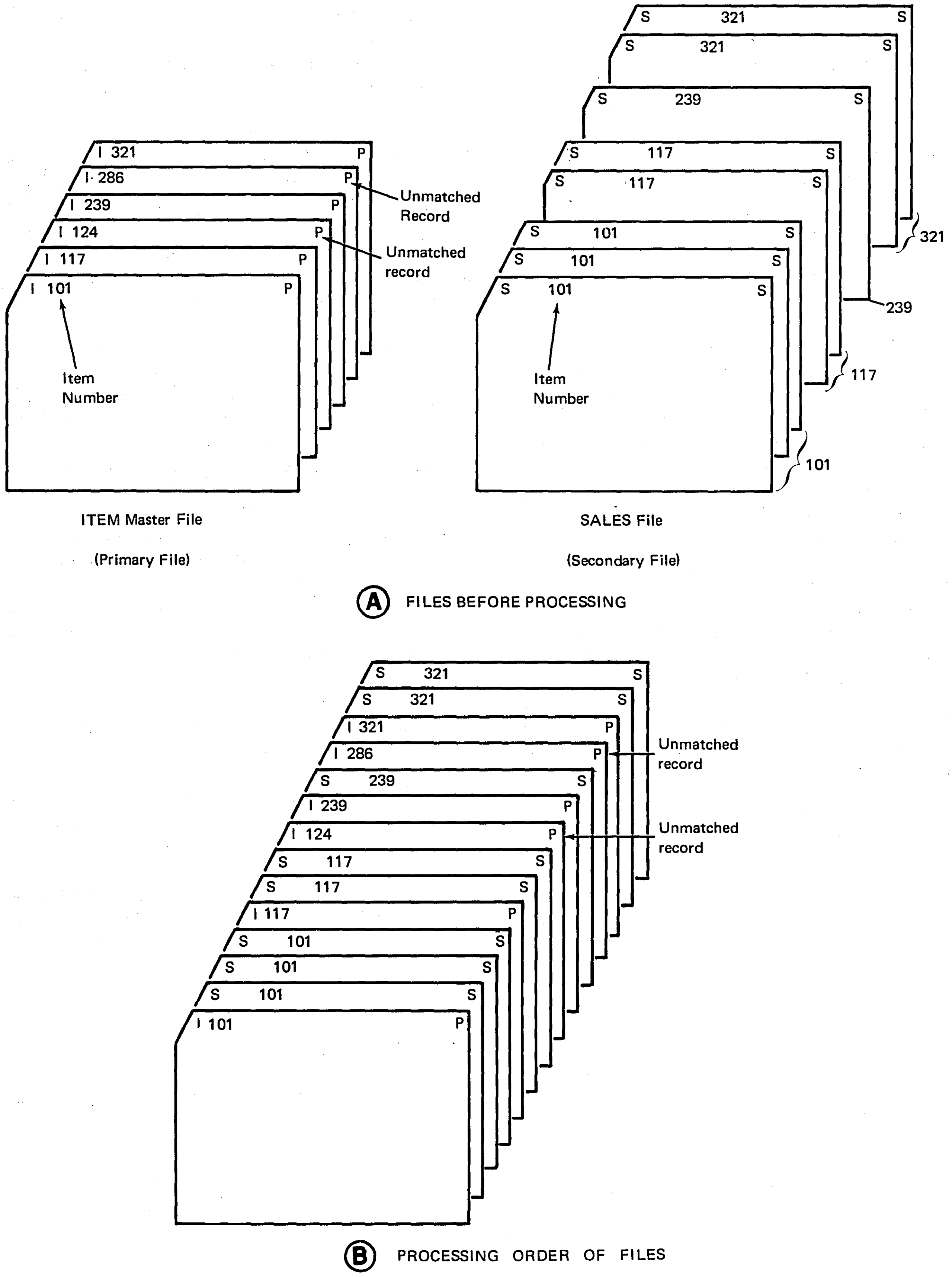
Following the ITEM record for 239, the SALES record for item 239 is processed. Then, as before, the unmatched ITEM record for item 286 is processed, since it is lower in sequence than item 321 on the next SALES record. To complete the processing, the primary file ITEM record for item 321 is processed, followed by the two sales records for item 321.

In summary, then, RPG II sets the MR indicator to off immediately before processing any unmatched records. MR is turned on before processing a record that has a matching record in the other file. After a record has been processed, the indicator remains as it was until it is set again immediately before the next record is processed.

Regardless of the records in a file, there is an easy way to determine if a matching record condition exists and if the MR indicator will be turned on:

1. When a primary file record is selected for processing, MR will be turned on if there is a secondary file record in the read area with the same match field.

2. When a secondary file record is selected for processing, MR will be turned on if the last primary file record processed had the same match field.



A FILES BEFORE PROCESSING

B PROCESSING ORDER OF FILES

Figure 6-18. Processing Files with Unmatched Records

Stacker Selection of Unmatched Card Records

When records from two files are assigned match fields, a record from one file may be processed, then a record from the other file, and then a record from the first file again. However, regardless of the order in which the records are processed, all cards which enter through the primary hopper of the MFCU or MFCM are automatically stacked in stacker 1. All cards entering through the secondary hopper end up in stacker 4 (stacker 5 on the MFCM Model A1).

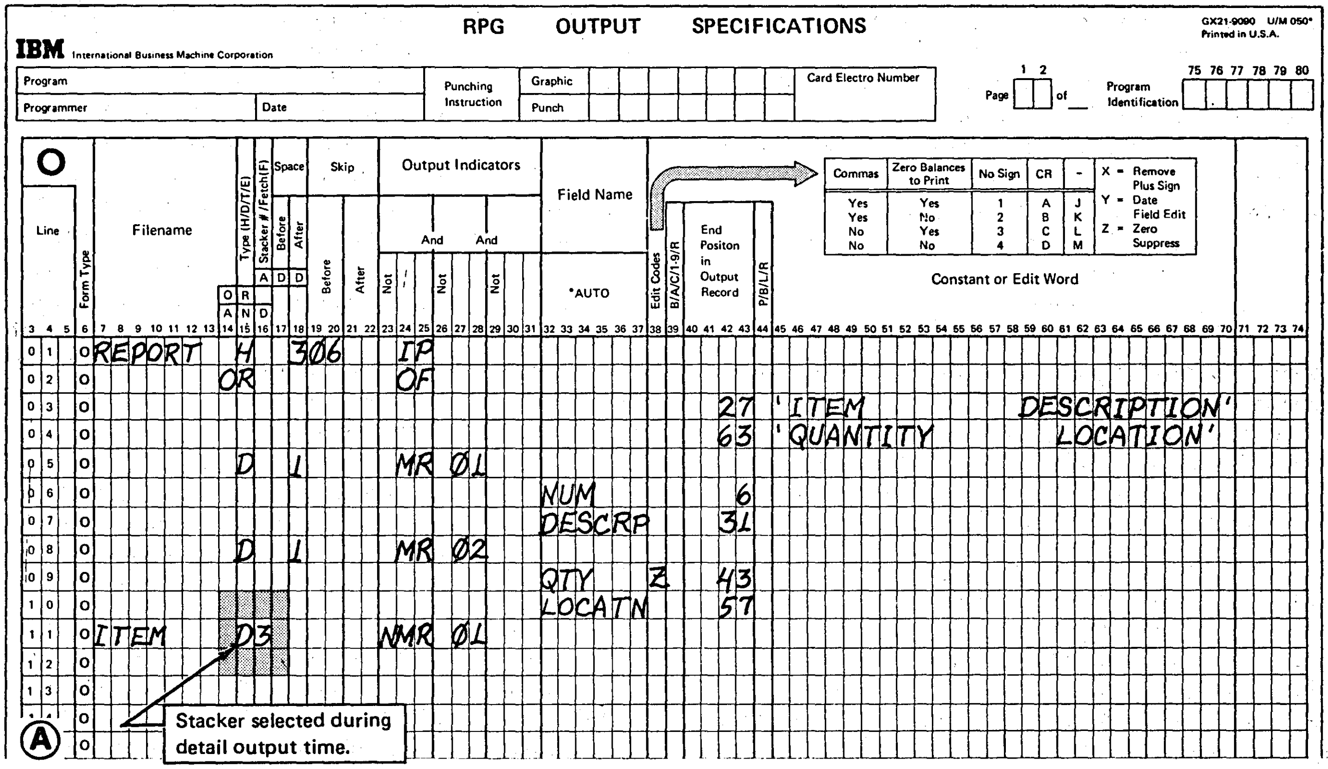
If there are unmatched records in either file, you can separate such cards from the others in that file by stacker selecting the unmatched cards into a different stacker. As you learned in the chapter *Card Output Operations,* cards from an input file can be selected into a different stacker by entering the number of the stacker on the Input sheet. However, this can be done only when input cards are to be stacker selected on the basis of record type.

Stacker selection of a card because it has no matching record in the related file must be specified in column 16 of the Output sheet (Figure 6-19, part A). However, an input file cannot be specified on the Output sheet. Therefore, the input file containing the card to be stacker selected (the card with no related record) must be defined as a combined file on the File Description sheet (Figure 6-19, part B). The combined file name can then be used on the Output sheet for stacker selection.

Notice on the Output sheet that stacker selection is specified for the combined file ITEM (line 11), since only the ITEM file contains unmatched records. The filename indicates from which file the cards will be output. Since a matching record condition cannot possibly exist for any unmatched records, MR must be off. Therefore, the stacker selection of records is conditioned by NMR (off).

Any record, conditioned for stacker selection by the MR indicator, should be specified as a detail-time record (see Figure 6-19, part A, column 15). Otherwise, the next record to be processed would be stacker selected instead of the record you want. This is because the detail-time processing of a card is done within one program cycle and the processed record automatically passes into the normal output stacker, if not stacker selected. The total-time operations for that same record are not performed until the next program cycle, after another record has been selected for processing. During total-time operations, although the status of the MR indicator is still set for the previously processed record, the only record available to be stacker selected is the card just selected for processing.

Whenever you wish to have a certain type of output based on a matching or not matching record condition, it can be important that a record identifying indicator be used with MR or NMR to condition the output. The record identifying indicator determines from which record type the information will be punched, stacker selected, or printed. For the printer output file, output can come from either the SALES file or the ITEM file (Figure 6-19, part A). Therefore, record identifying indicator 01 or 02 is specified to indicate which record type is to be printed (lines 05 and 08). Record identifying indicator 01 is entered on line 11 in addition to NMR so that stacker selection will occur for the ITEM file only when an ITEM record has been processed.



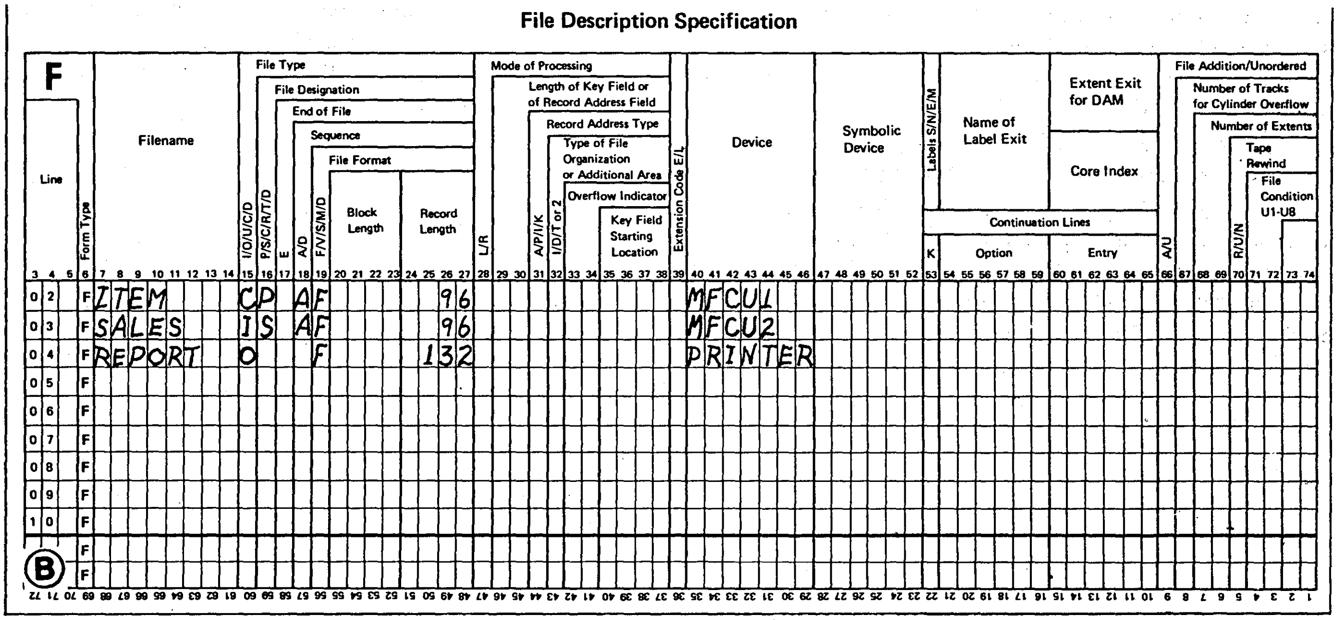
IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Stacker #	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	REPORT	H		3	06	IP				
02	O		OR				OF				
03	O									27	'ITEM DESCRIPTION'
04	O									63	'QUANTITY LOCATION'
05	O		D		1		MR Ø1				
06	O							NUM		6	
07	O							DESCRP		31	
08	O		D		1		MR Ø2				
09	O							QTY	Z	43	
10	O							LOCATN		57	
11	O	ITEM	D	3			NMR Ø1				
12	O										
13	O										
14	O										
	O										

Stacker selected during detail output time.



File Description Specification

Line	Form Type	Filename	File Type (I/O/U/C/D)	File Designation (P/S/C/R/T/D)	End of File (E)	Sequence (A/D)	File Format (F/V/S/M/D)	Block Length	Record Length	Device
02	F	ITEM	C	P		A	F		96	MFCU1
03	F	SALES	I	S		A	F		96	MFCU2
04	F	REPORT	O				F		132	PRINTER
05	F									
06	F									
07	F									
08	F									
09	F									
10	F									
	F									
	F									

Figure 6-19. Stacker Selection of Unmatched Records

MATCHING RECORDS: MORE THAN ONE RECORD TYPE IN A FILE

Match Fields in Different Locations in the Same File

Let's consider again the specifications necessary in assigning match fields for the weekly sales report. This time, however, suppose the SALES file used in producing the report contains two record types, charge records and cash records (see Figure 6-20). Both record types contain the item number, quantity, and location. In addition, charge records contain an account number, which cash records do not have. The cash sales records are identified by a C and an S in positions 1 and 2. Charge sales records contain an S in position 2, but no C in position 1. Furthermore, the match fields are in different locations on the two types of SALES records.

Recall the discussion of assigning the match fields for sequence checking within a file (see *Checking Sequence of Records Within a File*). When match fields are used in more than one record type in a file, the match field entries must either be specified one for each record type or in a field-record relation. The way they are coded depends on whether the match fields are found in different locations on each record type. If match fields are in different record locations, they may be coded either way. However, the ease of coding separate specifications for each record type may be more important to you than the specifications and storage often saved by using field-record relation.

In assigning entries for match fields which are to be used for matching records from two files, the same considerations must be made. The only difference is that, for matching records, you are concerned with two files, rather than one. You know that the match field entries on the Input sheet must be assigned *separately for each file*. Then, if one of the files has more than one record type containing the match fields, the record types *in that file* can be described separately or with field-record relation. If you use field-record relation, just be sure a particular match field field entry applies to all of the appropriate record types within the file.

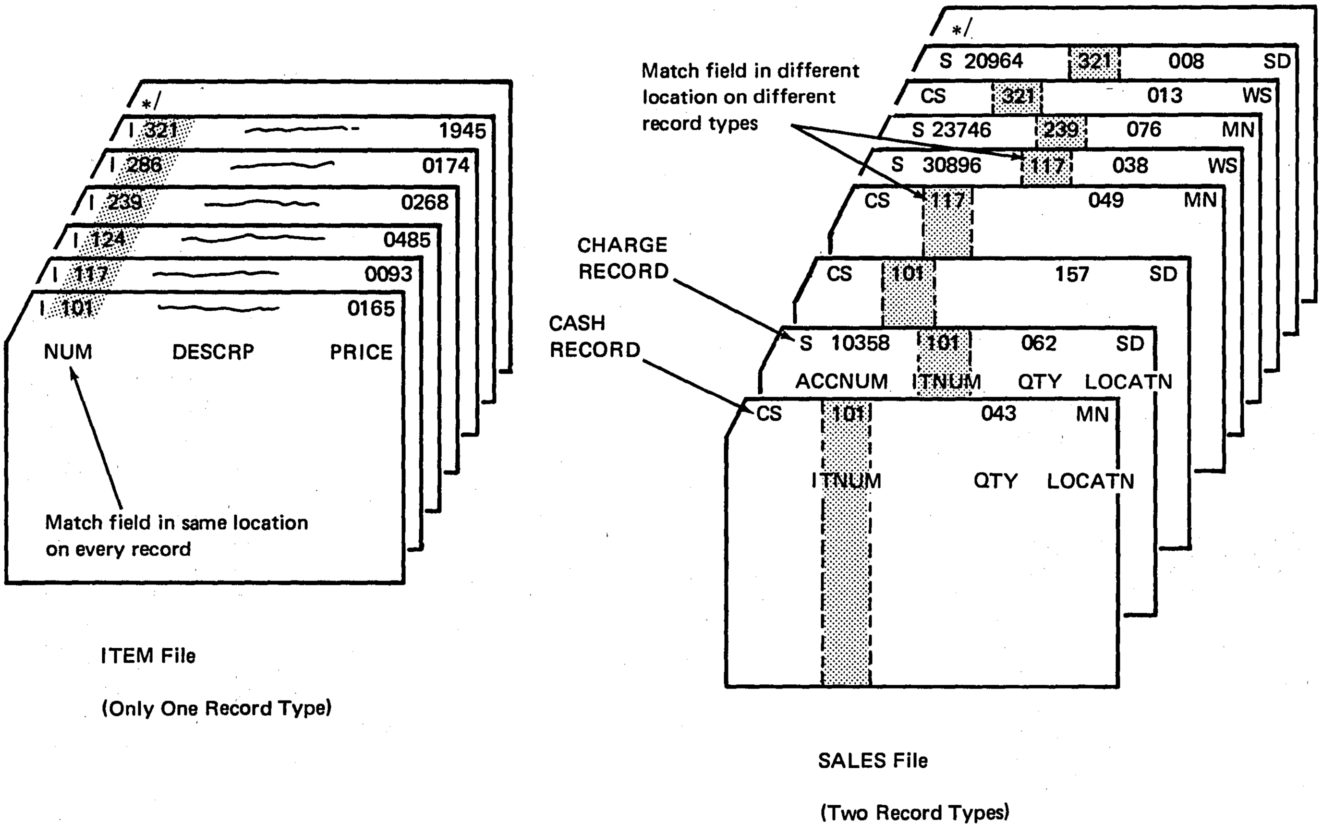


Figure 6-20. More than One Record Type in File Used for Matching

As shown in Figure 6-21, first the ITEM file is described as it was before (see Figure 6-16, part B). The M1 entry is assigned only once since there is only one record type in this file. The two record types in the SALES file have been described in an OR relationship, because most of the fields are in the same location on both types of SALES record. Note that all fields which are the same for both record types of the SALES file are specified first with no field-record relation entry. Then all the fields associated with the first type of SALES record only (02 for charge records) are specified, followed by those for the next record type (03 for cash records). The M1 entry must be assigned twice for the SALES file since the match field ITNUM is in different locations on the two record types.

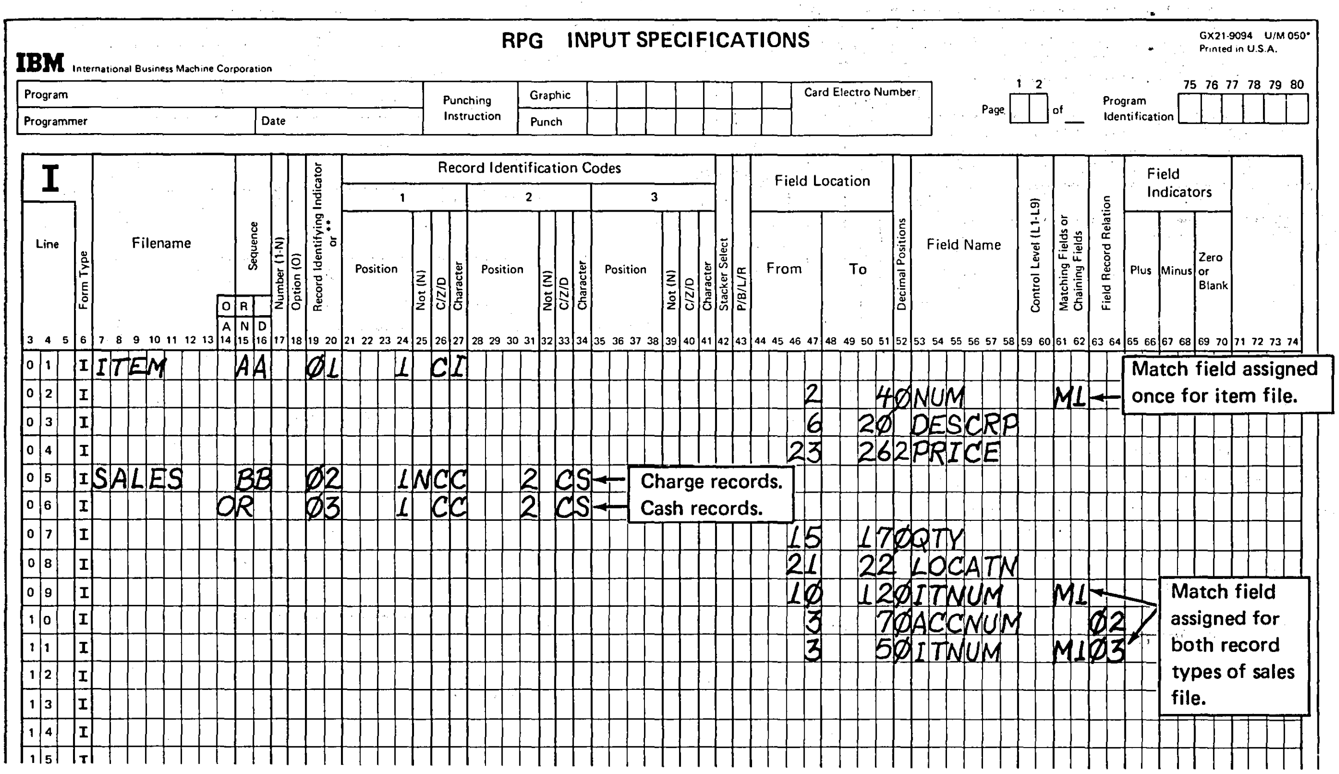
For all records of the same record type, the match fields (as well as all other fields) must necessarily be in the same location for every record *of that type.* However, match fields may be in different locations within the same file, providing there is more than one record type.

Although there is more than one record type in the SALES file, the order in which the records are selected for processing would be the same as explained previously for only one record type in each file. Before a comparison of match fields can be made between files, RPG II must first determine what type of record was read so it knows where the match field is located on that record.

There are some important restrictions on the use of match fields for matching records between files. All records for which match fields are specified must contain the same number of match fields. This applies to both files, regardless of how many record types are in either file. In the sales report program, only one match field (item number) was used. However, just as for sequence checking, up to nine match fields may be used to match records between files.

In assigning match fields to records, remember that for matching purposes, RPG II considers all match fields of the same value (M1-M9) to be in the same format, numeric or alphameric. Thus, if any of the match fields are defined as being numeric, RPG II looks at only the digit portion of all the other match fields with the same M1-M9 value, even though some may be alphameric fields.

If more than one match field is assigned, all the match fields in a record from one file must be equal to all the match fields in the record from the other file before a matching record condition exists. For example, assume that M1, M2, and M3 are match fields assigned to records in two files. In comparing records from each file, if the M1 and M2 fields are the same, but the M3 fields differ, the records do not match. If the files were specified to be in ascending sequence, the record with the lower value match field (M3, M2, M1 combined) would be selected for processing.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

```
01 IITEM    AA  ØL   1 CI
02 I                                        2   4ØNUM     M1
03 I                                        6  2Ø DESCRP
04 I                                       23  262PRICE
05 ISALES    BB  Ø2   1NCC   2 CS
06 I       OR    Ø3   1 CC   2 CS
07 I                                       15  17ØQTY
08 I                                       21  22 LOCATN
09 I                                       1Ø  12ØITNUM   M1
10 I                                        3   7ØACCNUM    Ø2
11 I                                        3   5ØITNUM   M1Ø3
12 I
13 I
14 I
15 I
```

Match field assigned once for item file.

Charge records.

Cash records.

Match field assigned for both record types of sales file.

Figure 6-21. Describing Record Types and Assigning Match Fields for Two Files

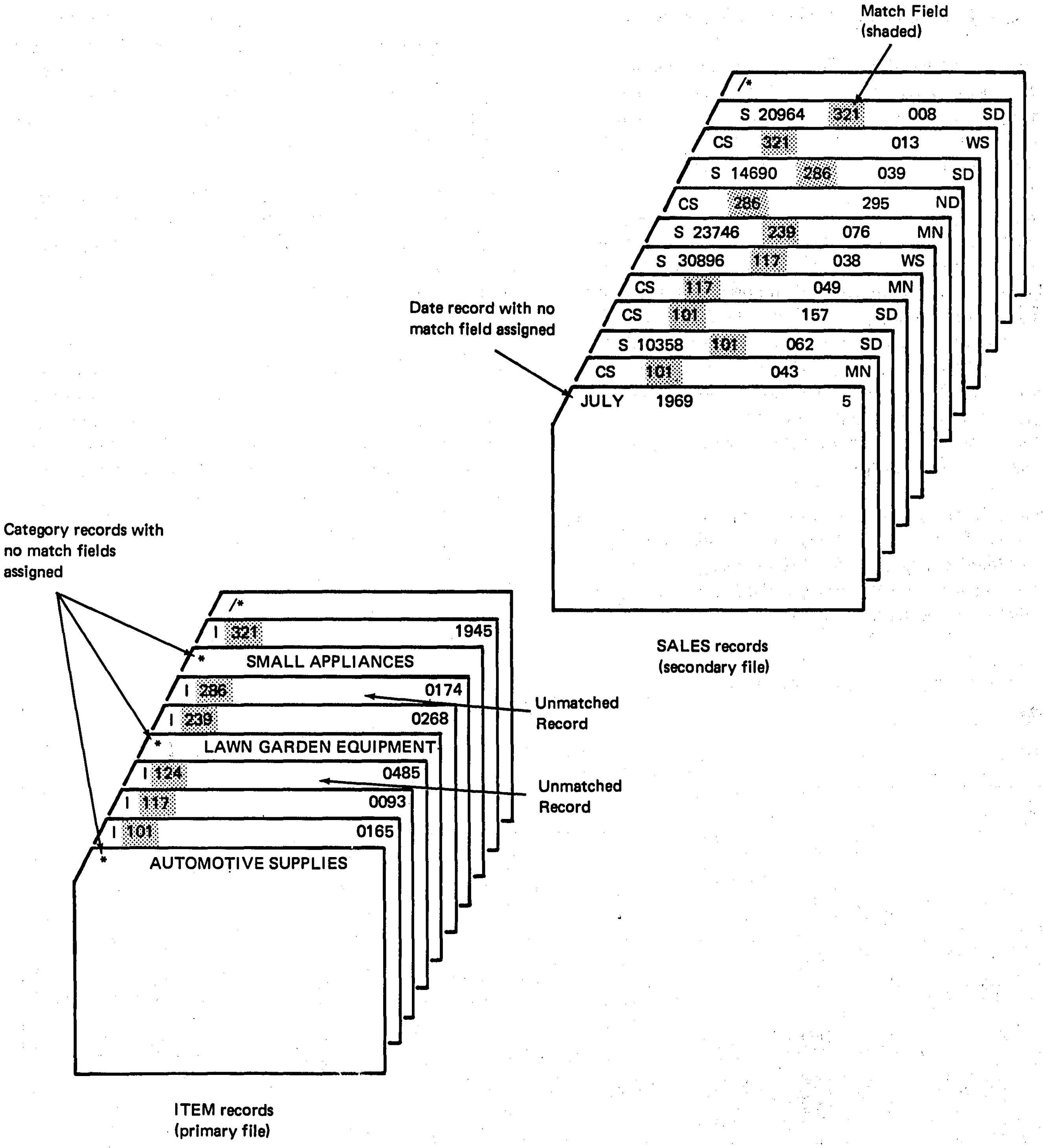


Figure 6-22. Files Containing Records Without Match Fields

Processing Records Which Do Not Have Match Fields

Up to now, all records in a file have been assigned match fields to be used in matching records from one file with those in the other file. Some files, however, may contain some records which are not to be matched; thus, such record types would not contain match fields.

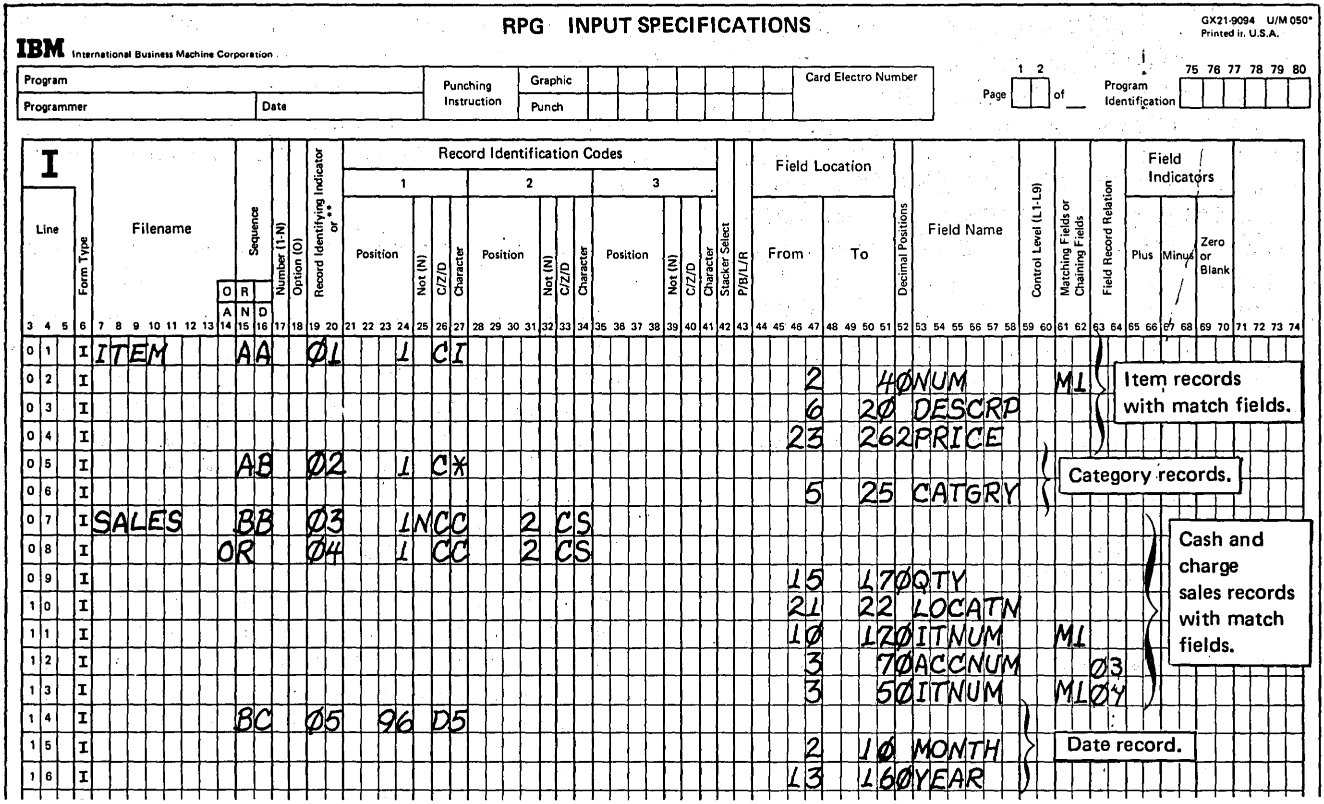
As you recall, the ITEM file was organized in ascending sequence according to item number. Item numbers are assigned such that items can be grouped according to a general category. All items assigned numbers from 100-199 are automotive supplies, those from 200-299 are lawn and garden equipment, 300-399 are small household appliances, and so on. In addition to the ITEM records for each item, then, the ITEM master file contains a record preceding each group, identifying the general category to which the items belong (Figure 6-22). These category records are distinguished by an * in position 1.

The SALES file also contains a record type which does not have a match field. In addition to the two types of sales records, one for cash sales and one for charge sales, there is a DATE record at the beginning of the SALES file, identified by 5 in position 96 (see Figure 6-22).

The two record types in the ITEM master file and the three record types in the SALES file would be described as shown in Figure 6-23. On the basis of the record identifying codes, one of the record identifying indicators which are assigned to the record types (positions 19-20) will turn on to indicate which record type was read.

Notice that the input specifications (Figure 6-23) assign a match field to only some of the record types present in the two files. The category records from the ITEM file and the date record from the SALES file are not to be matched with records from the other file.

Since match fields are used to determine the order in which records are processed, when will the category and date records be processed? Records for which no match fields are specified are processed immediately as they are read. To understand the order of processing and when the MR indicator is on, then, let's discuss how RPG II logic operates during the program cycles for this program.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)
01	I	ITEM	AA			Ø1	1		C	I											
02	I														2	4	Ø	NUM		M1	
03	I														6	2Ø		DESCRP			
04	I														23	26	2	PRICE			
05	I		AB			Ø2	1		C	*											
06	I														5	25		CATGRY			
07	I	SALES	BB			Ø3	1	N	C	C	2		C	S							
08	I	OR				Ø4	1		C	C	2		C	S							
09	I														15	17	Ø	QTY			
10	I														21	22		LOCATN			
11	I														1Ø	12	Ø	ITNUM		M1	
12	I														3	7	Ø	ACCNUM			Ø3
13	I														3	5	Ø	ITNUM		M1	Ø4
14	I		BC			Ø5	96		D	5											
15	I														2	1Ø		MONTH			
16	I														13	16	Ø	YEAR			

Field Indicators: Plus, Minus, Zero or Blank

Item records with match fields.

Category records.

Cash and charge sales records with match fields.

Date record.

Figure 6-23. Describing Records With and Without Match Fields Assigned

At the beginning of the first program cycle, the first record is read from each file. RPG II then decides if match fields are specified for both records. As mentioned, a record for which no match fields are assigned is automatically selected for processing before the record in the other file.

In this case, neither record in the read area has match fields. Therefore, which should be processed? Without match fields, the two records cannot be compared to see which is lower in sequence. When both records do not have match fields, the record from the primary file has priority and, thus, is selected for processing.

Since match fields are not assigned to this record type, there can be no matching record in the other file. Therefore, immediately before a record without match fields is processed, the MR indicator is turned off.

For the next program cycle, the ITEM record for item 101 is read to replace the category record processed. The date record from the secondary file is still in the read area. Since the secondary file record has no match fields, no comparison is made, and the date record is automatically processed. Once again, the MR indicator is turned off right before processing a record without match fields.

The remaining records would be processed in the order shown in Figure 6-24. When records with match fields are in the read area, the record with the lowest match field is selected for processing. If both match fields are the same, the record from the primary file is always selected. Of course, if one of the records has no match field, it is processed immediately after the record it follows, regardless of which file it is in. As shown in Figure 6-24, there are matching SALES file records for the ITEM file record with a match field value of 286. Since a record without match fields (the category record for SMALL APPLIANCES) follows this ITEM record in the primary file, the record without match fields is processed before the SALES records which match ITEM record 286. You should be aware that this processing order may be undesirable in your program.

MATCHING RECORDS: WHEN ALL RECORDS IN ONE FILE HAVE BEEN PROCESSED

When you are using two or more input-type data files in a program, end of file is always reached in one file before all the cards in the other file have been processed. This is because the matching record function can only select one card at a time for processing. Furthermore, each file often contains a different number of records.

Usually, in multifile processing, you want the program to continue until all records from all files have been processed. RPG II logic will do this. The last record indicator (LR) is not turned on until the last record in the last file is read.

Let's go over the last few program cycles in the example last presented. At a particular point in the program, there will be two records left in the primary file (item 321 and a /* record) and three records in the secondary file which have not been processed (two for item 321 and a /* record).

At the beginning of a program cycle, the ITEM record for 321 and the first SALES record for item 321 would be in the read area. Since both match fields are the same, the primary file record is processed, with the MR indicator turned on before processing.

The next primary file record, an end of file (/*) record, is then read. In a single file program, reading a /* record tells the program there are no more data records to process. The last record (LR) indicator is then turned on causing total time operations to be performed, and then the program ends.

In multifile processing, however, reading a /* record from one file does not necessarily mean that all records from all files have been processed. Thus, the program must determine if end of file has previously been reached in the other file(s). In this case, the SALES file still contains three unprocessed records. Therefore, reading the /* record from the primary file merely indicates that there are no more records from that file to process. The remaining data records in the SALES file are processed, one at a time, in the order in which they are read.

When the end of file record from the secondary file is read, the program determines if end of file has been reached in the other file. Since it has, the LR indicator is then turned on. In this program, there are not total calculations or total output to be performed; thus, the program is ended.

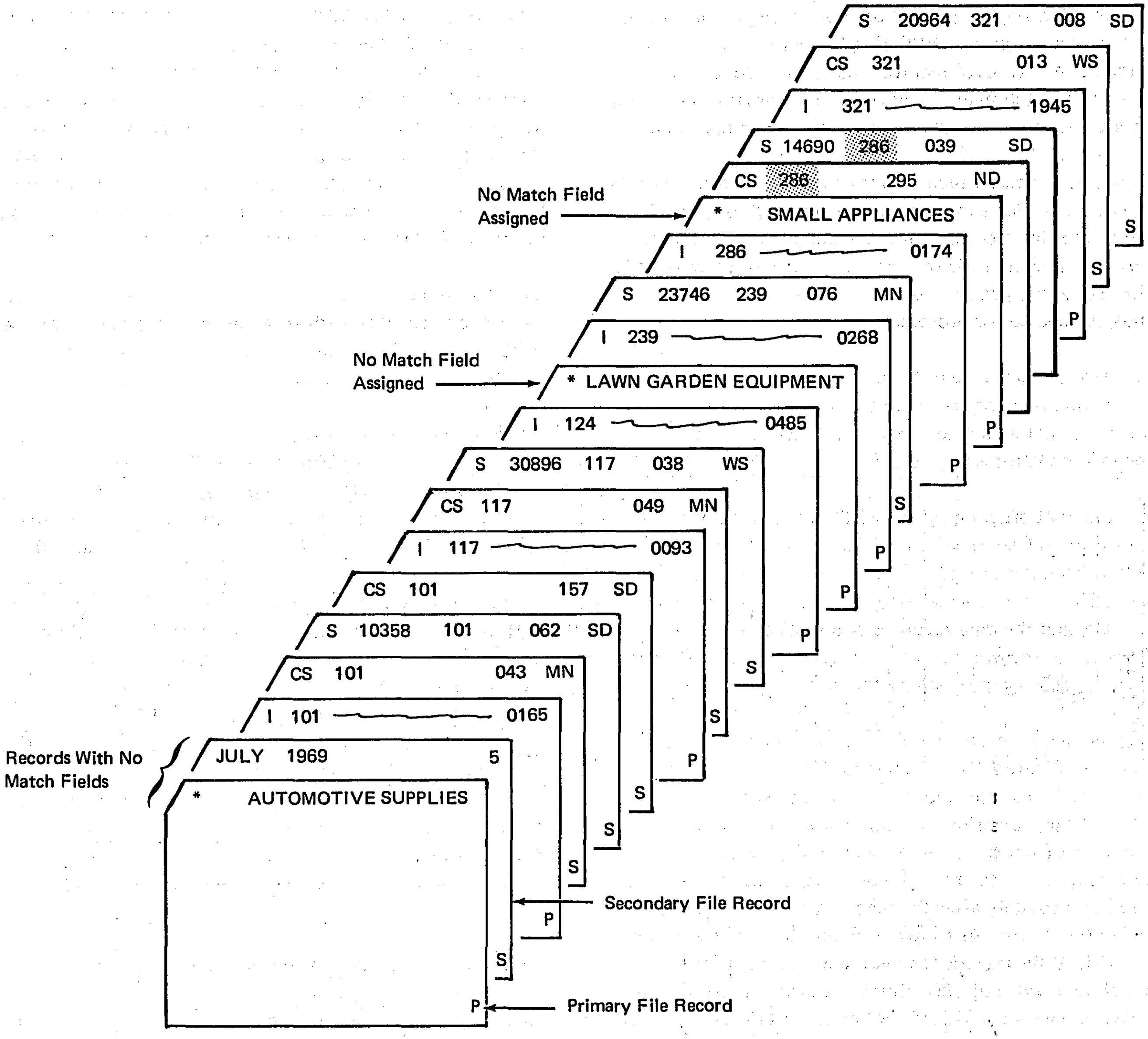
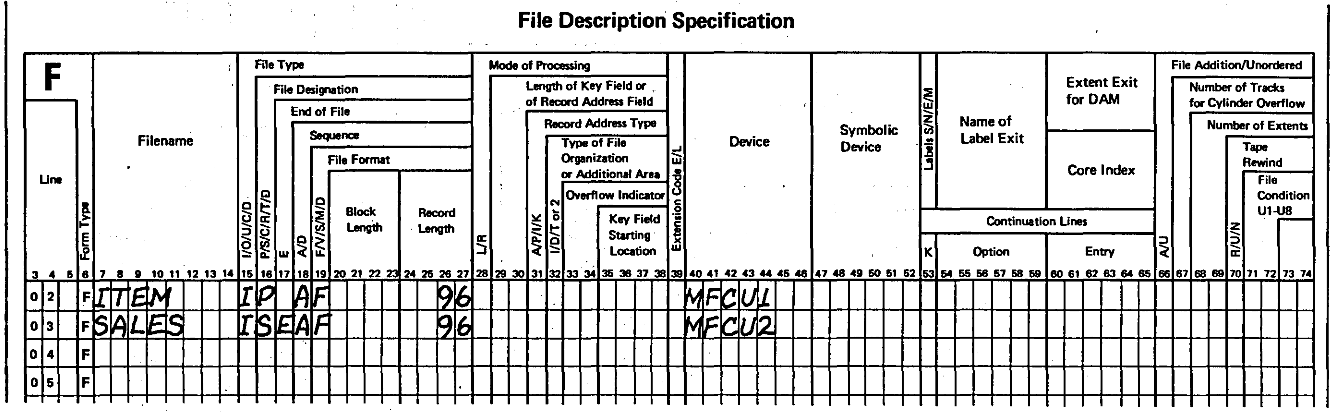


Figure 6-24. Processing Order of Records with No Match Fields

In other multifile programs, you may want the program ended when a certain file has been completely processed, although there are records left in another file. As you know, by entering an E in column 17 of the File Description sheet you can specify which file is to end the program. For example, say the ITEM and SALES files contain the records shown in Figure 6-25. Since both files are in ascending sequence, once all SALES records have been processed, there wouldn't be much point in processing the remaining ITEM master records for which no sales were made. Thus, an E (end of file) entry could be specified for the SALES file.

You should be aware, however, that in a matching records program, specifying the end of file entry for a particular file will not always cause processing to end immediately when the last record of that file is processed. If the file which has not been completely processed contains records which match the last record processed from the completed file, the matching records will be processed before the program is ended. Furthermore, processing will continue for any records without match fields, until the next record with a match field is read.



File Description Specification

Line (3-5)	Form Type (6)	Filename (7-14)	File Type (15)	File Designation (16)	End of File (17)	Sequence (18)	File Format (19)	Block Length (20-23)	Record Length (24-27)	Device (40-46)
02	F	ITEM	I	P		A	F		96	MFCU1
03	F	SALES	I	S	E	A	F		96	MFCU2
04	F									
05	F									

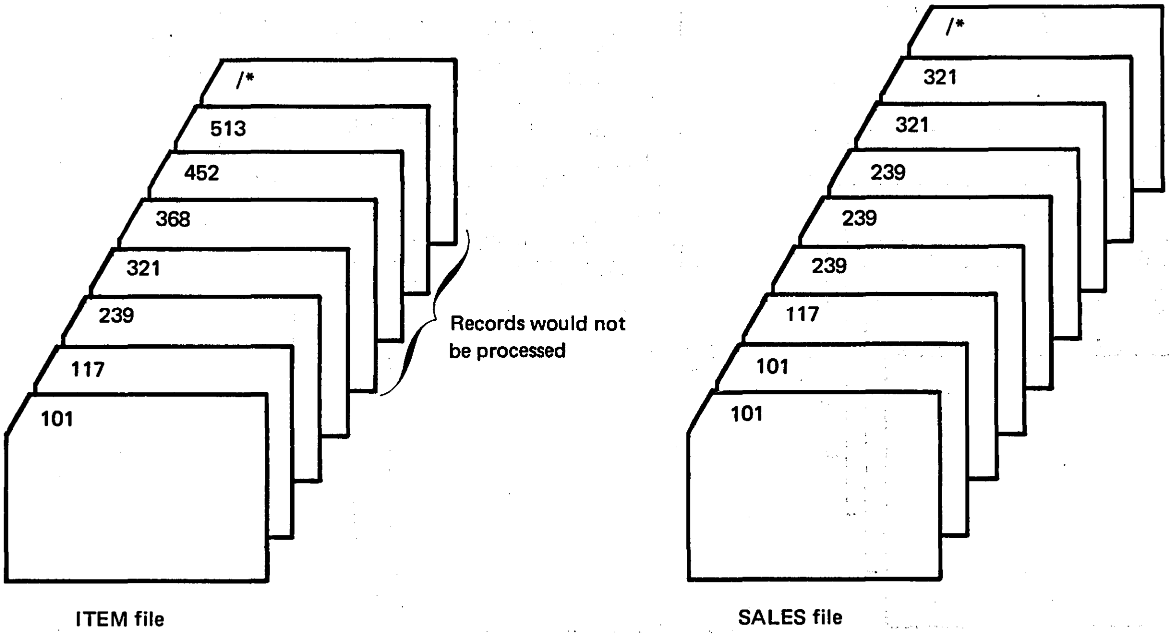


Figure 6-25. Specifying When Processing is to Stop in a Matching Records Program

Assume the secondary SALES file contains the records shown in Figure 6-26. Following each group of sales records for a particular item is a record which gives the total sales for that item. The total records do not have match fields and thus are selected for processing as soon as they are read.

An end of file entry is assigned to the primary ITEM file. Since both files are in the same sequence, once all ITEM master records are processed, any sales records which still remain in the SALES file must be either records which match the last primary record processed, records for which there is no valid item number, or total records which are not to be matched.

When the /* record in the primary file is read, the program checks the other file. The two SALES records for item 321 match the last primary file ITEM record processed. Thus, they are automatically processed. The next record in the SALES file, a total record without match fields, will cause the LR indicator to be turned on, and the program will end. However, if an end of file entry was not assigned to the primary ITEM file, or if the end of file entry was assigned to the secondary SALES file, all records in the SALES file will be read and processed.

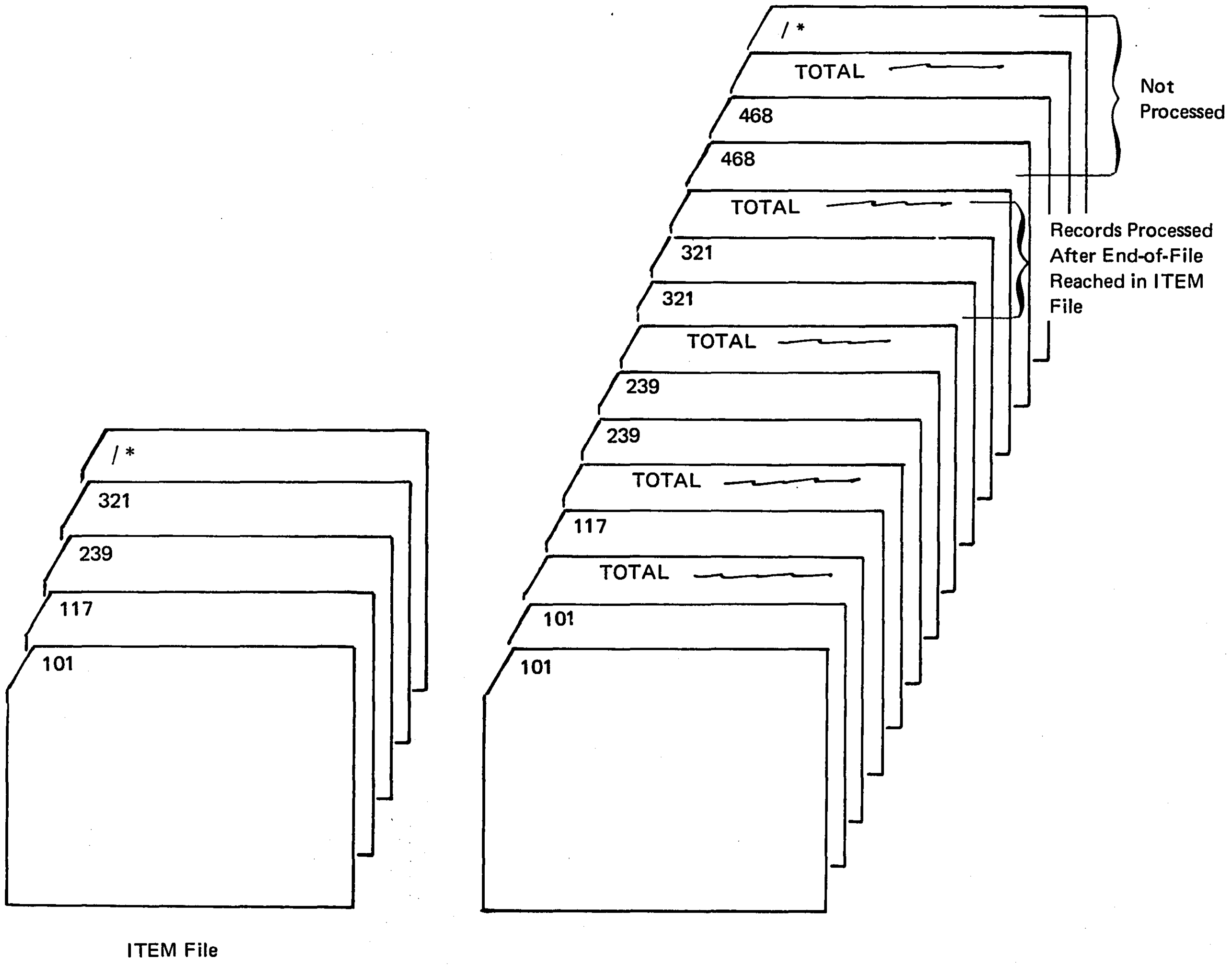


Figure 6-26. Processing a Matching Records Job After End of File

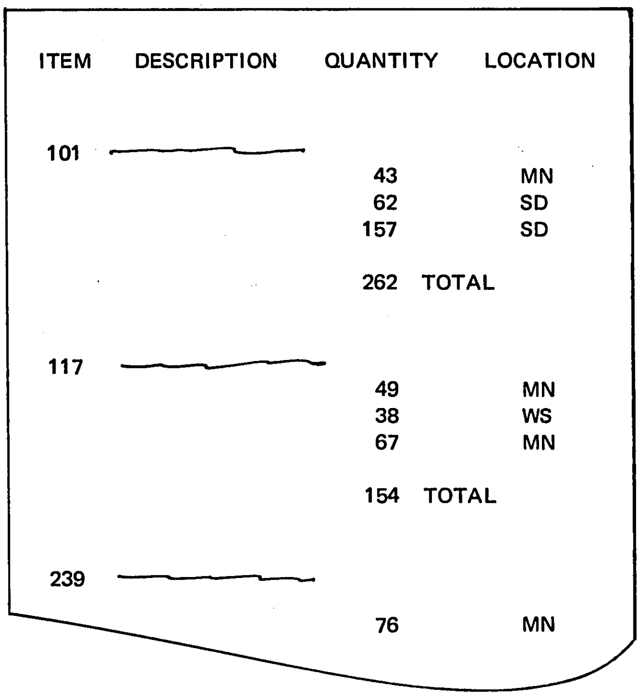
USE OF MATCH FIELDS AND CONTROL FIELDS IN THE SAME FILE

In the previous example, the ITEM and SALES files were matched according to an item number field so that individual sales could be printed under the item description. Suppose you also wish to have the sales for each item totaled and printed, as shown in Figure 6-27.

To perform total operations on a group of records, it is necessary that control fields be assigned to distinguish one group from the next. For this program, the same item number field used for matching the records would be specified as a control field on the Input sheet (Figure 6-28). Although your files may contain both match fields and control fields, there is no relationship between the two functions performed. Even if the same fields on a record are used as both match and control fields, the only similarity is that the same data will be used in performing each function.

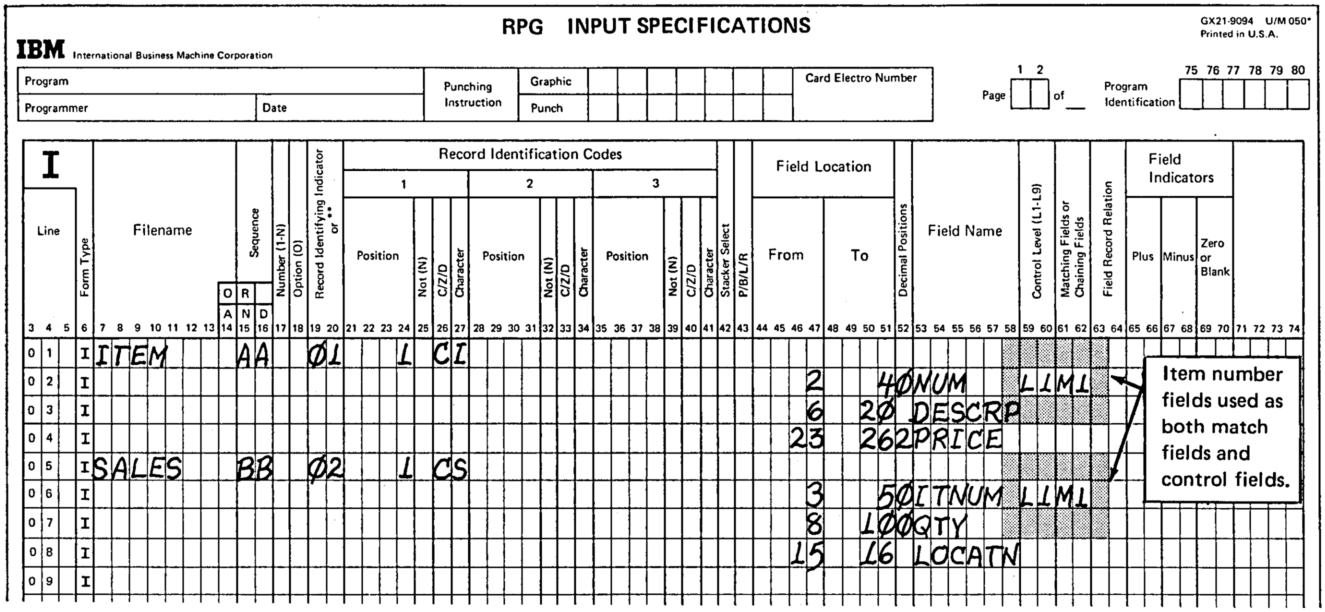
Match fields are checked first to determine from which file the next record is to be processed (see Figure 6-29). In effect, the matching record function is creating one file for processing by merging the data from two files. (Of course, after processing, the two files are automatically separated again into the appropriate stackers.)

In addition, comparison of the match fields determines whether the MR indicator will be turned on or off for processing of the selected record. As discussed previously, the MR indicator is to be on for processing of a primary file



```
ITEM    DESCRIPTION     QUANTITY    LOCATION

101

                           43           MN
                           62           SD
                          157           SD

                          262   TOTAL

117

                           49           MN
                           38           WS
                           67           MN

                          154   TOTAL

239

                           76           MN
```

Figure 6-27. Printing Totals for Matching Records



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Programmer | Date

I

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Code 1 Position	Not (N)	C/Z/D	Character	Code 2 Position	Not (N)	C/Z/D	Character	Code 3 Position	Not (N)	C/Z/D	Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Plus	Minus	Zero or Blank
01	I	ITEM	AA			Ø1	1		C	I																				
02	I																				2	4	Ø	NUM	L1	M1				
03	I																				6	2Ø		DESCRP						
04	I																				23	26	2	PRICE						
05	I	SALES	BB			Ø2	1		C	S																				
06	I																				3	5	Ø	ITNUM	L1	M1				
07	I																				8	1Ø	Ø	QTY						
08	I																				15	16		LOCATN						
09	I																													

Item number fields used as both match fields and control fields.

Figure 6-28. Assigning the Same Field as a Match Field and as a Control Field

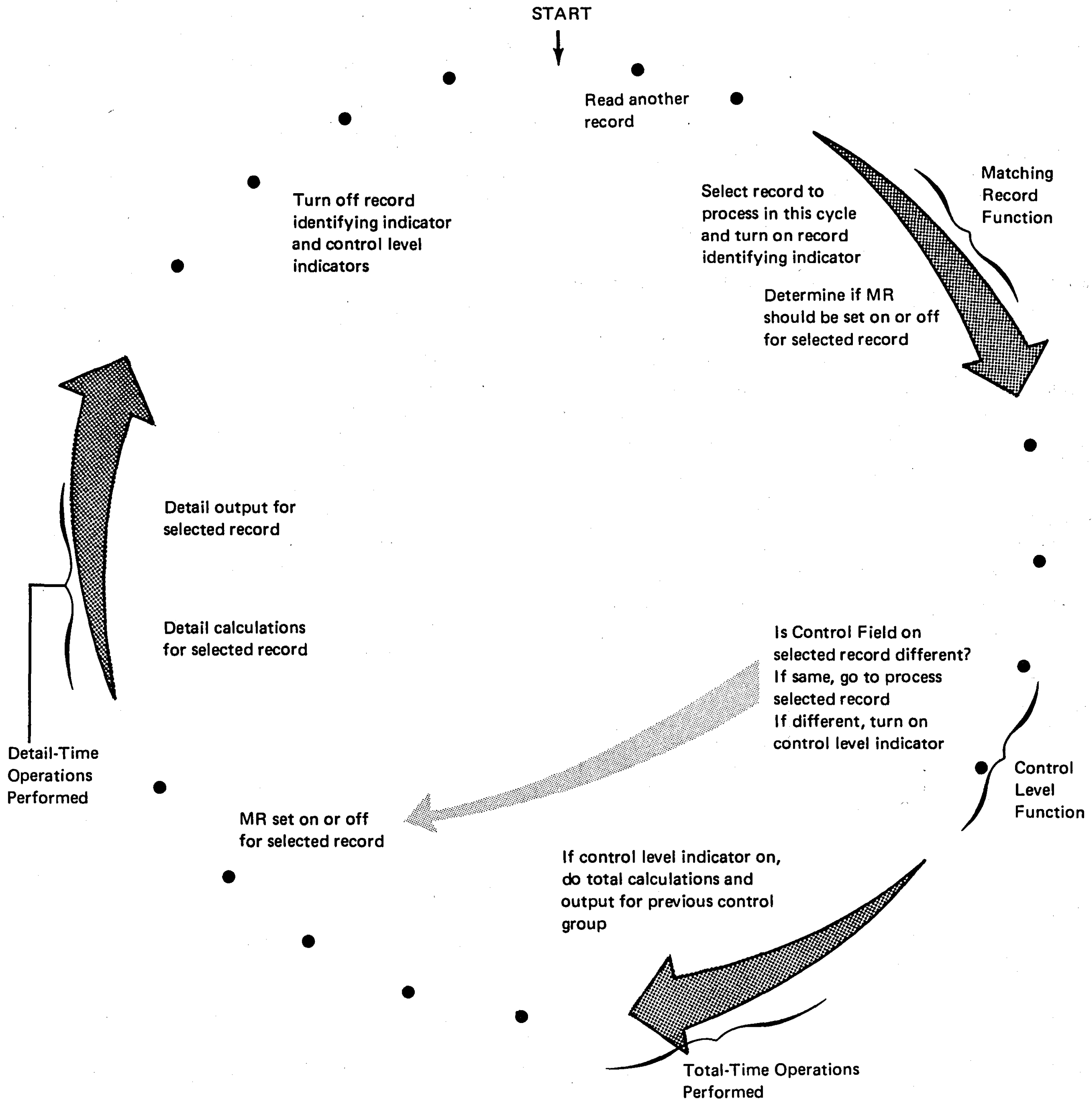

START
Read another record
Select record to process in this cycle and turn on record identifying indicator
Matching Record Function
Determine if MR should be set on or off for selected record
Is Control Field on selected record different? If same, go to process selected record If different, turn on control level indicator
Control Level Function
If control level indicator on, do total calculations and output for previous control group
Total-Time Operations Performed
MR set on or off for selected record
Detail-Time Operations Performed
Detail calculations for selected record
Detail output for selected record
Turn off record identifying indicator and control level indicators

Figure 6-29. RPG II Logic of Obtaining Totals for Matching Records

record only if there is a matching record in the secondary file. Likewise, if a secondary file record is selected, MR is to be on for its processing only if a matching record from the primary file has already been processed. If an unmatched record is selected, MR is set off, of course. At this point, however, the matching record function only determines how the MR indicator is to be set; the status of MR is not actually changed yet.

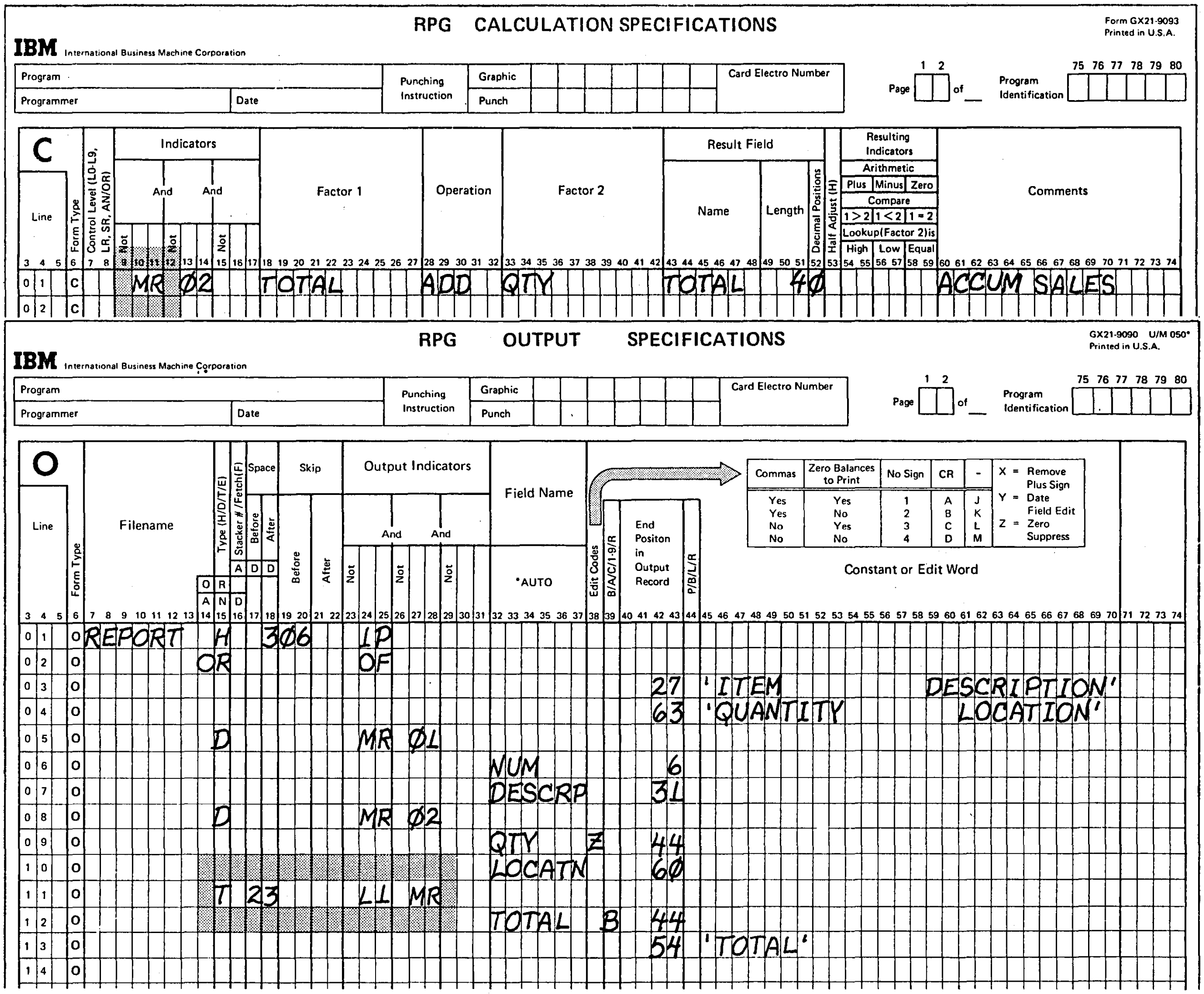
Once the matching record function has selected the next record to be processed, the control level function then considers the records as one file in determining if a control break has occurred. This check is made before the selected record is processed and, thus, before the MR indicator is set for the record just selected (see Figure 6-29). The control field (item number) on the selected record is checked to see if it is different from that on the previously processed record. If it is the same, no totals are to be printed;

so the MR indicator is set, the QTY accumulated into TOTAL, and the individual record is printed. However, if the item number is different from that on the previously processed record, this means that all individual sales for the last item number have been printed. The change in the control field causes a control level indicator (L1 was assigned) to be turned on. Any total calculations and total output (operations conditioned by the control level indicator) are then performed for the previous control group. In this case, the total of all sales for the previous item are printed.

When total operations are performed, the selected record has not been processed yet. Therefore, during total time, MR is still set for the previously processed record. Once total operations are completed, MR is then set for the selected record (the first record in the next control group) so it can be processed.

Whenever a control field changes (control break), a control level indicator is automatically turned on. Furthermore, whenever a control level indicator is on, total operations will be performed. In a matching records program, however, there may be times when a control break occurs and you don't want total operations to be done. Thus, you must specify that total operations are to be performed only under certain conditions.

For this program, sales are to be added and the total printed *only* if there is an ITEM master record with matching SALES records for that item. In such a case, MR would have been set on for the last SALES record of the group. MR would still be on, then, when the control break occurs. Thus, detail operations to accumulate sales should be performed whenever MR is on and the total operations to print the TOTAL should be conditioned to be performed only with *L1 and MR* on (Figure 6-30).



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	Resulting Indicators	Comments
01	C		MR Ø2	TOTAL	ADD	QTY	TOTAL	4	Ø			ACCUM SALES
02	C											

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	REPORT	H		3	Ø6		1P						
02	O		OR					OF						
03	O											27		'ITEM DESCRIPTION'
04	O											63		'QUANTITY LOCATION'
05	O		D					MR Ø1						
06	O								NUM			6		
07	O								DESCRP			31		
08	O		D					MR Ø2						
09	O								QTY	Z		44		
10	O								LOCATN			6Ø		
11	O		T	2	3			L1 MR						
12	O								TOTAL	B		44		
13	O											54		'TOTAL'
14	O													

Figure 6-30. Controlling Performance of Total Operations in a Matching Records Program

Since there can be unmatched records in either file, you can have an ITEM record with no matching SALES records or invalid SALES records for which there are no matching ITEM master (see Figure 6-31). Although a control break would occur following the processing of the unmatched records, you don't want total operations to be performed. The MR indicator would be off for the unmatched records. Therefore, if you condition total time output specifications with L1 *and MR* on, total time would be bypassed for such cases.

DETERMINING WHETHER FILES SHOULD BE PRIMARY OR SECONDARY

A basic question arises in any multifile processing application: Which file should be designated the primary file and which is to be the secondary file? Since almost all multifile programs involve the use of match fields to determine processing order, an understanding of matching record logic is generally essential for determining which file should be the primary file.

Since files vary among applications, we can't state absolutely that a certain file is always to be primary or secondary. However, now that you understand the basic matching records concepts, the following points may help you to make the best file designation.

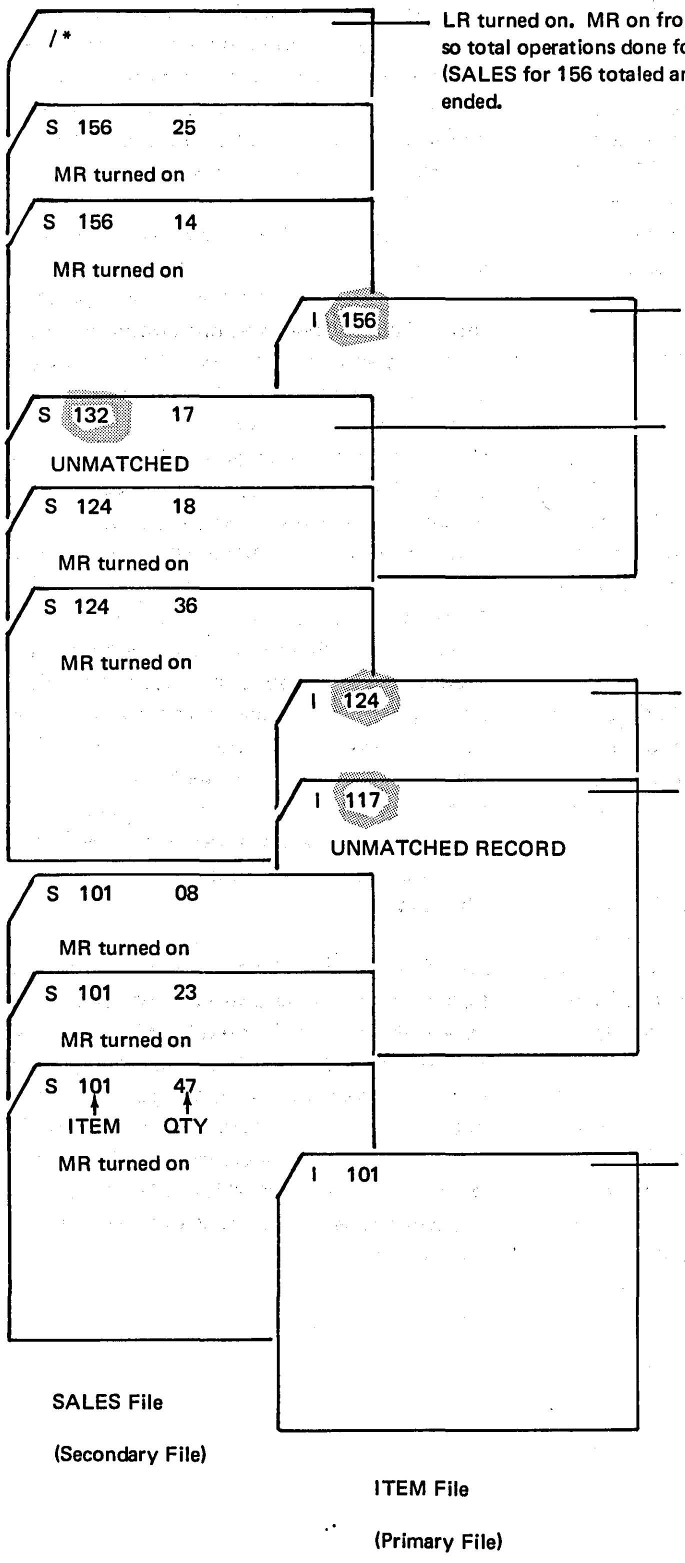
If the match fields on records from two or more files are compared and found to be the same, the primary file record is always selected. Furthermore, the primary file would be given priority if none of the available records contained match fields. In general, then, if all match conditions are the same, the primary file record is the first processed.

With this idea in mind, we can say that if data from file A must be available to the program before file B can be processed, the file containing the necessary data (A) should be designated as the primary file. For example, say you have two files to process to do a customer billing application. The customers' names and addresses are recorded in one file while the customers' transactions are in another file. A name and address is to be printed on a bill before a customer's charges and credits. The name and address record must be available for output and, thus, must be processed before the related transaction cards. In this case, the name and address file should be specified as the primary file.

Let's consider another case in which one file is to be used only as input while the other file is to be used for both input and output. In other words, the program requires one input file to be read and one combined or update file to be both read and written or punched. Usually, information from the input file would be necessary to calculate the data to be output to the combined or update file. Or, perhaps the actual input file data is to be output to the combined or update file. Either way, the combined or update file should be specified as the secondary file, so the data from the primary input file is available before the output is done.

In general, the type of data which must be available before processing another file is permanent record information, such as would be found in a master file. For instance, an employee's hourly wage from a master file is necessary before the computer can use the number of hours worked from an employee detail file to calculate net pay. Therefore, a master file is often the primary file, while a detail or transaction file is the secondary file. Of course, this is only a guideline and should not be considered the rule for all situations.

As a final point, if the first record in one of the files *must* be the first record processed in the program, you must make sure that this record would be selected. To do this, first decide which file you think should be primary and which secondary, according to the suggestions presented. Then, determine which record would be selected first in the program by the RPG II logic of matching records. If the record which would be selected is not the record which must be processed first, the primary and secondary file designation should be switched.



LR turned on. MR on from previous record so total operations done for previous control group. (SALES for 156 totaled and printed) Job is then ended.

L1 turned on. MR off from previous record so total operations skipped. MR then turned on for this matching record.

L1 turned on. MR on from previous record so SALES for 124 are totaled and printed. MR then set off for this unmatched record.

L1 turned on. MR off from previous record so total operations skipped. Then MR turned on for this matching record.

L1 turned on. MR still on for previous record so SALES for 101 are totaled and printed. MR then set off for this unmatched record.

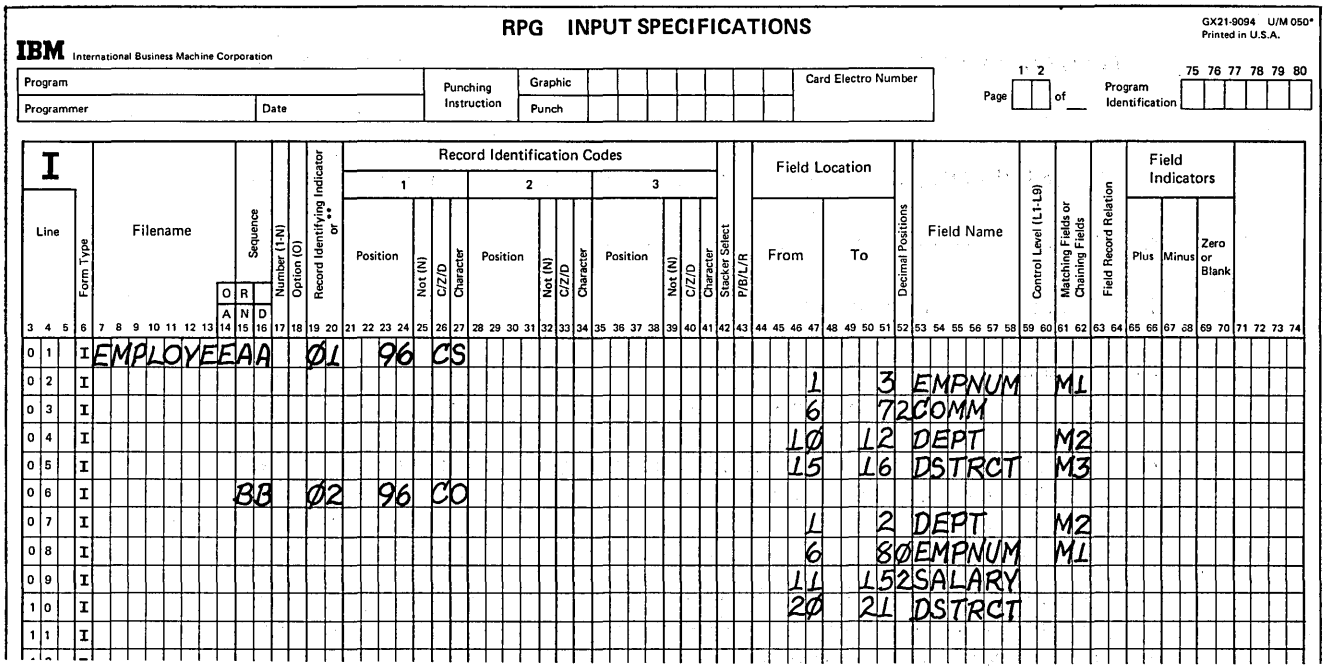
L1 turned on automatically for first record. Total operations are skipped in first program cycle, however. MR then set on for this matching record.

Figure 6-31. Use of MR Indicator to Determine Whether Total Operations are to be Performed

1. What are match fields used for when assigned to a single file?

2. If a record type has two match fields, which match field is assigned M1 and which is assigned M2?

3. Referring to the following Input sheet, indicate the errors in assigning match fields and the reasons the specifications are in error.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Plus	Minus	Zero or Blank
01	I	EMPLOYEE	AA			Ø1	96		C	S										
02	I										1	3		EMPNUM		M1				
03	I										6	7	2	COMM						
04	I										1Ø	12		DEPT		M2				
05	I										15	16		DSTRCT		M3				
06	I		BB			Ø2	96		C	O										
07	I										1	2		DEPT		M2				
08	I										6	8	Ø	EMPNUM		M1				
09	I										11	15	2	SALARY						
10	I										2Ø	21		DSTRCT						
11	I																			

4. Must match fields be assigned to all record types in a file?

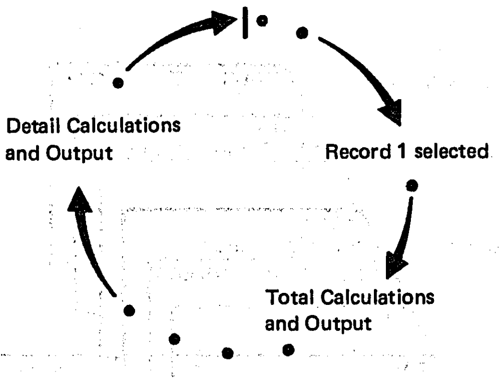
5. Given the following data about a file named EMPLOYEE, code the input specifications which describe the record types in an OR relationship:

Record type 01 identified by the character A in position 96.
Record type 02 identified by the character B in position 96.

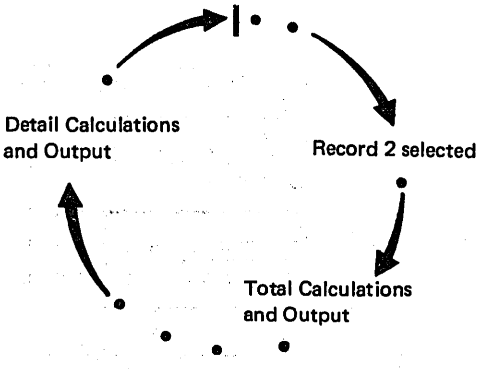
Field Name	*Record Positions*	*Record Type*
NAME	5-15	record type 02 only
DEPT	1-4	both record types
CODE	8-9	record type 01 only
ADDRES	16-30	record type 02 only
EMPNUM (match field)	42-45	both record types

6. What are match fields used for when assigned to multiple input, update, or combined files?

Refer to the following program cycles to answer questions 7 and 8:



PROGRAM CYCLE 1



PROGRAM CYCLE 2

7. When is the MR indicator set on and off for the first record selected for processing?

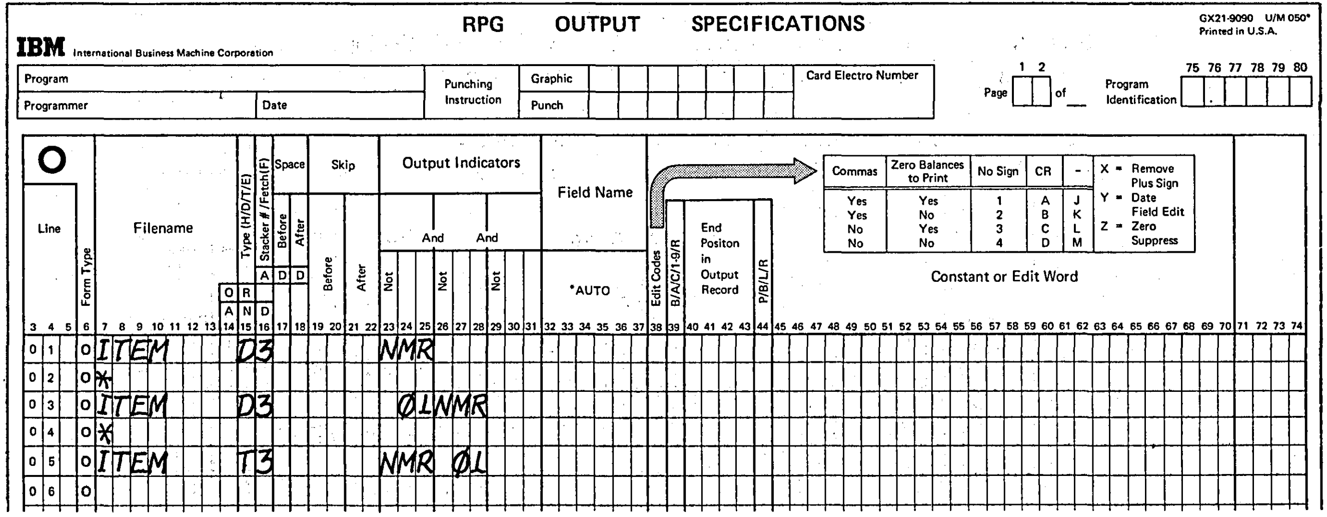
8. When is the record identifying indicator for record 1 set on and off?

9. Define an unmatched record.

10. If a primary file record is selected for processing, what condition must be met for the MR indicator to be set on for that record?

11. If a secondary file record is selected for processing, what condition must be met for the MR indicator to be set on for that record?

12. Assume an ITEM file read from the MFCU or MFCM contains two record types. Which specification line on the following Output sheet is correct in order to stacker select unmatched records of record type 01 into stacker 3? What are the other two stacker selection specifications incorrect?


IBM International Business Machine Corporation

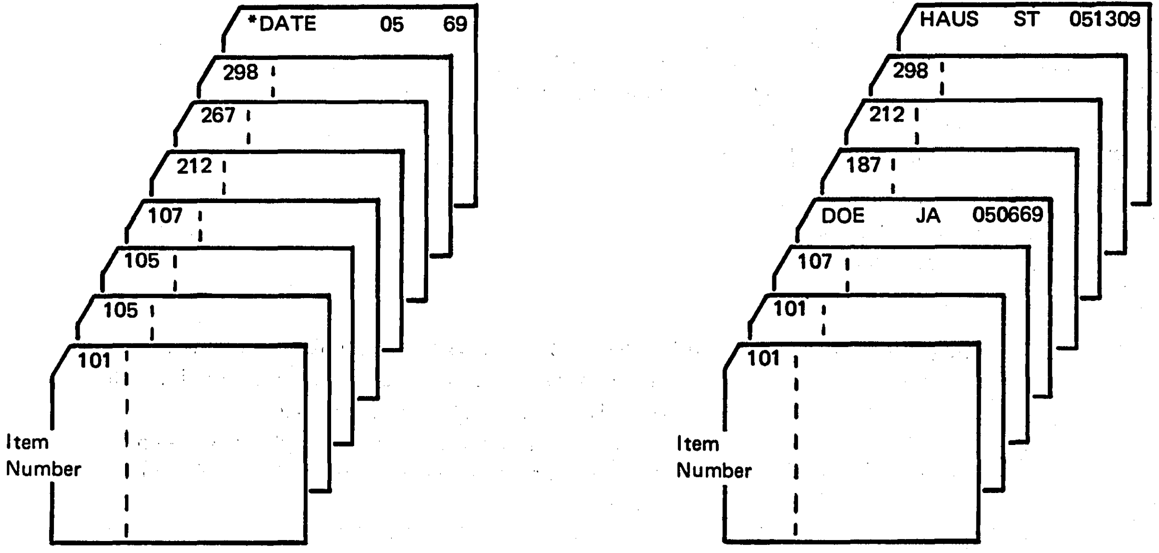
RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page of
Program Identification

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker #/Fetch(F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators Not (23)	(24-25)	Not (26)	(27-28)	Not (29)	(30-31)
01	O	ITEM	D	3					N	MR				
02	O	*												
03	O	ITEM	D	3						ØL	N	MR		
04	O	*												
05	O	ITEM	T	3					N	MR		ØL		
06	O													

13. An item number field is assigned as a match field for processing the two files shown. Identify the matching records (records for which MR is on), the unmatched records, and the records with no match field assigned.



Primary File

Secondary File

14. Referring to the files shown in question 13, specify the order in which the records would be processed.

15. What does an E in column 17 of the File Description sheet indicate? What does an A or D in column 18 indicate?

16. What do the following input specifications cause to happen?

IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program		Punching Instruction	Graphic	
Programmer	Date		Punch	

Card Electro Number

Page 1 2 of ___ Program Identification 75 76 77 78 79 80

I

							Record Identification Codes														Field Location							Field Indicators		
							1				2				3															
Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Stacker Select	P/B/L/R	From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Plus	Minus	Zero or Blank
3 4 5	6	7 8 9 10 11 12 13	14 15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70
0 1	I	SLSMAN	AA			Ø3	96		C	M																				
0 2	I																				1	5		REGION	L1	M1				
0 3	I	COMM	BB			Ø5	96		C	C																				
0 4	I																				85	89		REGION	L1	M1				
0 5	I																													

Review Problem

The data processing department is to prepare a weekly labor distribution report showing the total number of hours worked by each employee, as well as the total number of hours worked by all employees within a department. Therefore, the report must be organized by department number and by man number within a department.

Two input files are necessary to provide data for the report:

a. PMSTER is a master payroll file containing three types of records:

   - Date record – first record in file.
   - Department name records.
   - Employee master records.

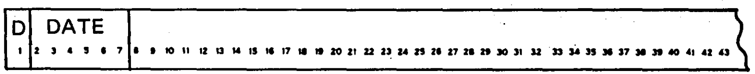
   The employee records are in ascending sequence by department number and by man number within a department. A department name record appears within the file immediately before the group of employee records for that particular department.

b. LABOR is a file of daily records containing the number of hours an employee worked. Since each employee's daily hours are recorded on a separate record, there will be more than one LABOR record for each employee.

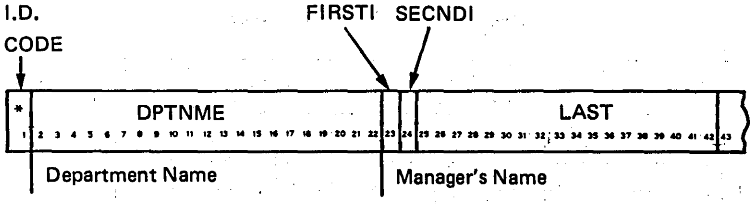
   The LABOR file is also in ascending sequence by department number and by man number within a department.

PMSTER File – 3 Record Types

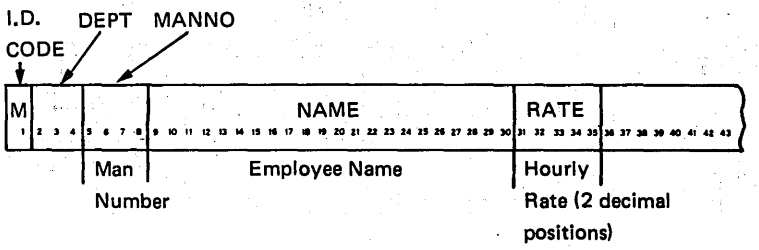
Date Record Layout



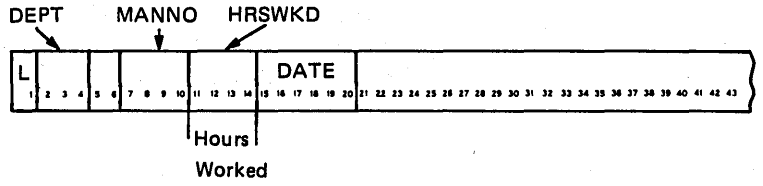
Department Name Record Layout



Employee Master Record Layout



LABOR File – 1 Record Type



In processing the two files, there should be an employee master record related to all LABOR records. In fact, more than one LABOR record will match the same employee master record from the PMSTER file. However, it is possible that the LABOR file contains the following unmatched records:

- Records with errors in match fields.

- Records for new employees for whom employee master records have not yet been created.

If you read the LABOR file from the MFCU or MFCM, unmatched LABOR records should be stacker selected into stacker 3. Processing should end when the last record from the LABOR file has been processed.

For this program, see if you can code the following:

1. File description specifications (read the files from MFCU, MFCM, or DISK).

2. Input specifications.

3. Output-format specifications necessary to stacker select unmatched LABOR cards using the MFCU or MFCM.

Answers To Review 6

1. Match fields are used to sequence check cards within a file.

2. The higher entry, M2, is assigned to the more important field.

3. Reasons for specification errors:

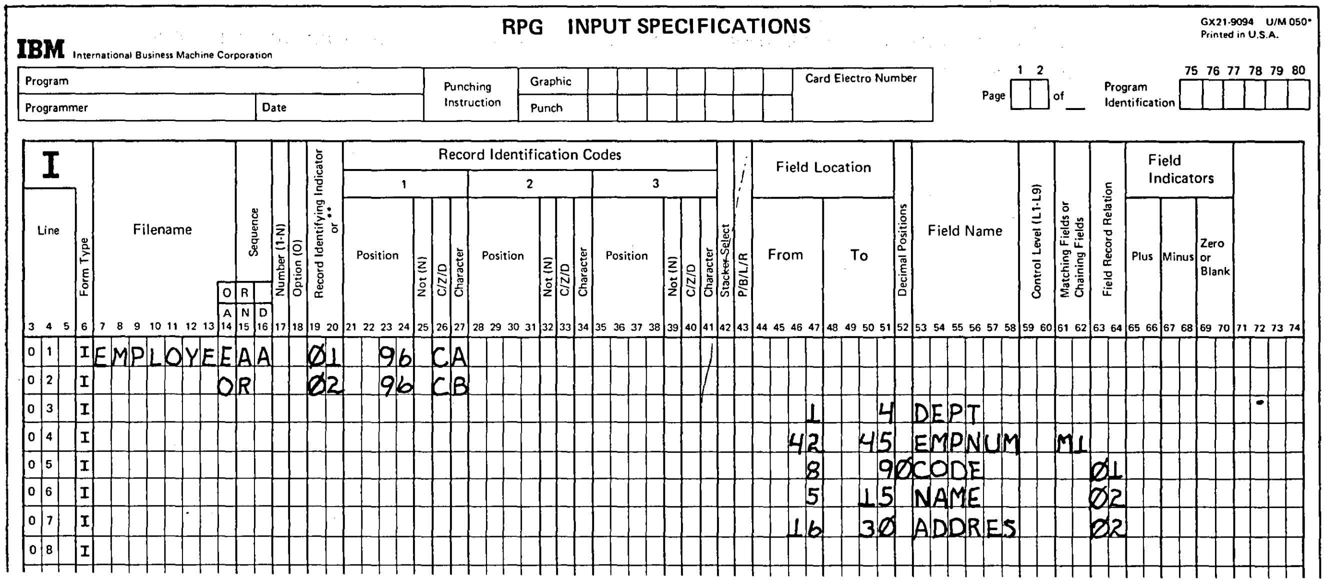
   Line 07 – Match fields of the same level must be the same length. Also, fields having the same name must be the same length, regardless if they are assigned as match fields or not.

   Line 08 – Match fields of the same level must be in the same format (alphameric or numeric) *if the fields have the same name.* Also, fields with the same name must be in the same format (alphameric or numeric), regardless if they are assigned as match fields or not.

   Line 10 – Every record type assigned match fields must contain the same number of match fields.

4. No. Remember that record types without match fields are selected before record types with match fields.

5. All fields which are common to all record types must be described first. All fields related to a particular record type are then described together as a group. Following the common fields, fields related to either record type 01 or those related to record type 02 may be described next, as follows:



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Program

Programmer

Date

Punching Instruction: Graphic / Punch

Card Electro Number

Page □□ of __

Program Identification

Line	Form Type	Filename	Sequence (OR/AND)	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank
01	I	EMPLOYEE	AA			Ø1	96		C	A																				
02	I		OR			Ø2	96		C	B																				
03	I																				1	4		DEPT						
04	I																				42	45		EMPNUM		M1				
05	I																				8	9	Ø	CODE			Ø1			
06	I																				5	15		NAME			Ø2			
07	I																				16	3Ø		ADDRES			Ø2			
08	I																													

6. Match fields are used for two purposes:
   a. Sequence checking the cards in each file
   b. Matching records between the two files to determine the order of processing

7. MR set on between total time and detail time operations *during program cycle 1.*
   MR set off between total time and detail time operations *during program cycle 2.*

8. Record identifying indicator set on immediately after record is selected during program cycle 1.
   Record identifying indicator set off following detail time operations during program cycle 1.

9. An unmatched record is a record in either input file which does not have a matching record (same match field) in the other input file.

10. There must be a record in the secondary file with the same match field(s) as the record in the primary file.

11. The match field(s) on the secondary file record must be the same as the match field(s) on the last primary file record processed.

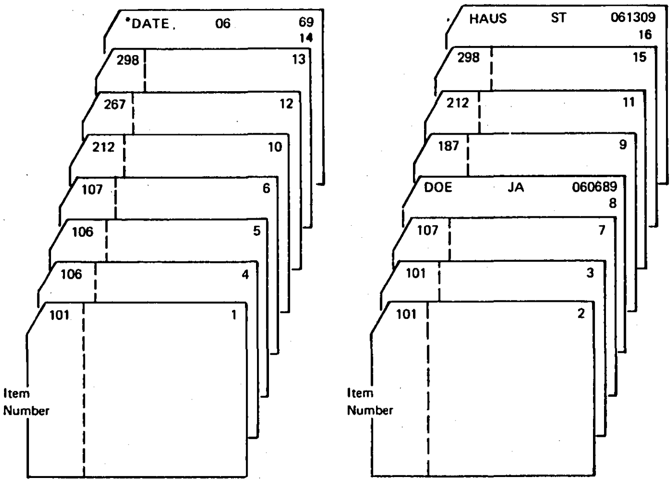
12. Line 03 is a correct specification for stacker selection of unmatched records.
    Line 01 is incorrect because there is no record identifying indicator specifying the type of record to be stacker selected.
    Line 05 is incorrect because stacker selection of a record on the basis of a matching or not matching record condition should be done at detail time, not at total time.

13. Matching records are records from both files containing item numbers 101, 107, 212, and 298.

    The three unmatched records from the primary file contain item numbers 105, 105, and 267. The one unmatched record in the secondary file contains item number 187.

    The *DATE record from the primary file and the two name records from the secondary file are records with no match fields assigned.

14. The records would be processed in the following order:



Primary File

Secondary File

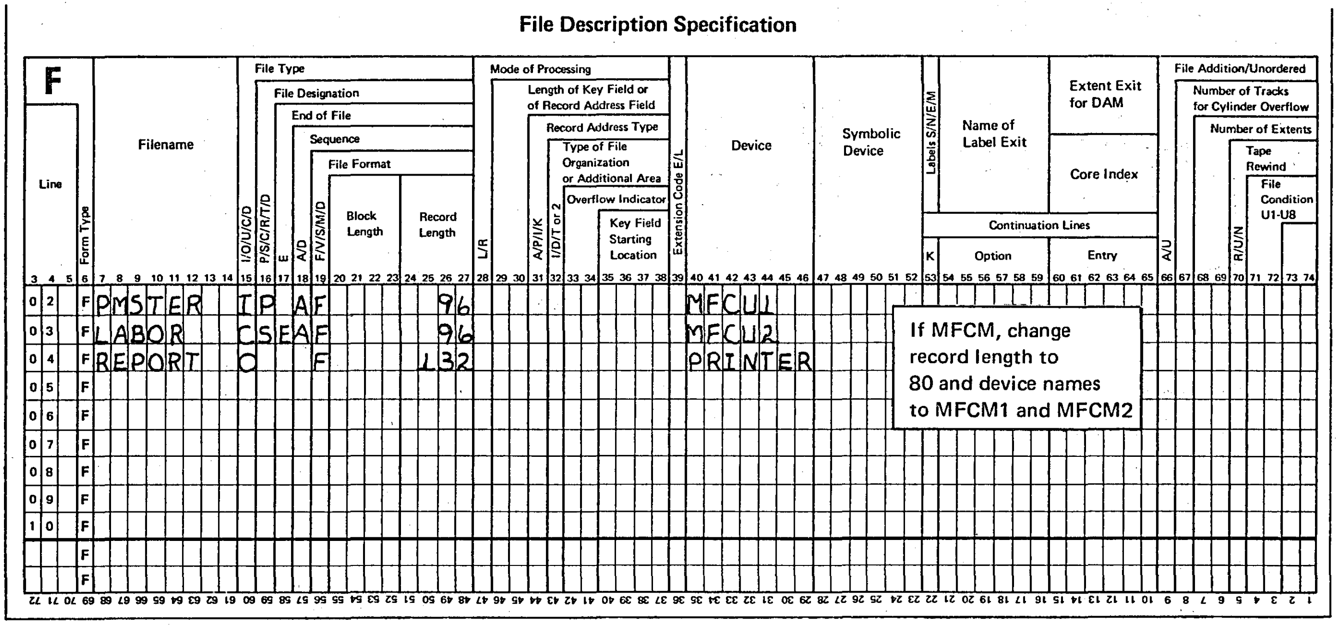
15. An E means the program should check that file for end of file. If end of file is reached (a /* record read), usually processing of records is stopped. However, if end of file is reached in a matching records program, the matching records and records with no match fields from the other file are processed before the program is ended.

    An A or D entry indicates that an input file is in ascending or descending sequence according to the match field(s) on the records.

16. First, the REGION match fields are compared to determine if there is a matching record condition and, thus, which record to process. Next, the contents of the REGION field are checked on the record selected to determine if a control break has occurred and, thus, if L1 should be set on.

Solution to the Review Problem

1. File Description Specifications
   a. MFCU or MFCM files:

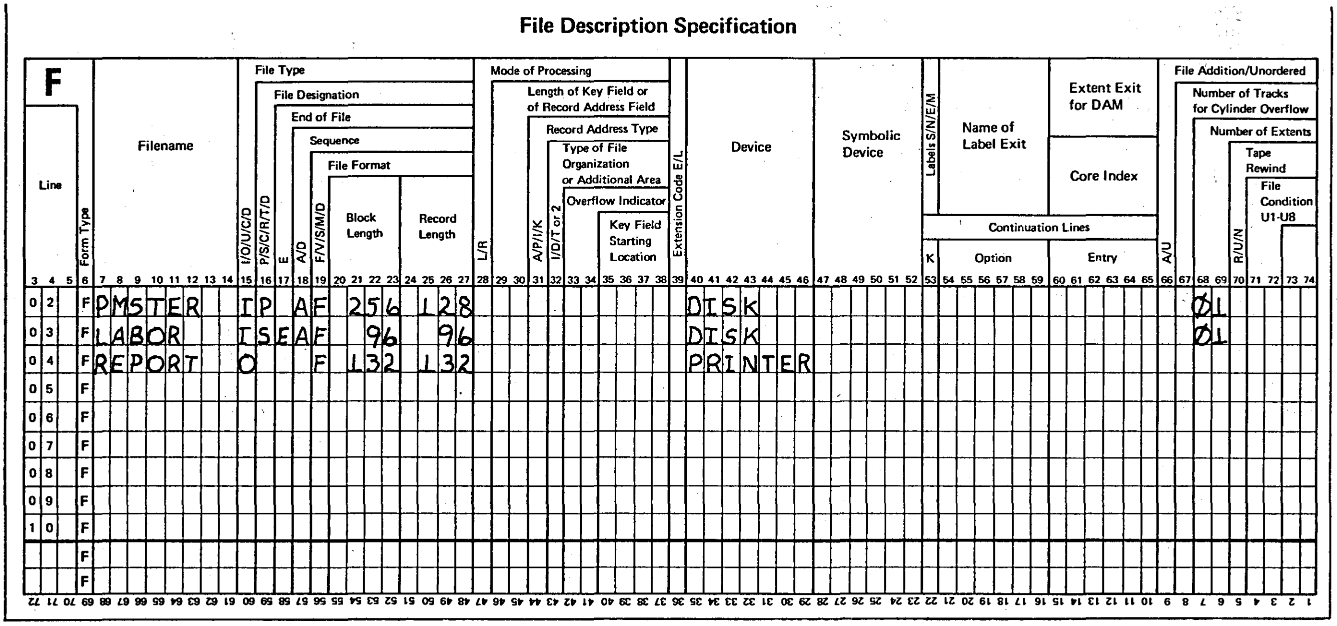


File Description Specification

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V/S/M/D	Block Length	Record Length	Device
02	F	PMSTER	I	P		A	F		96	MFCU1
03	F	LABOR	C	S	E	A	F		96	MFCU2
04	F	REPORT	O				F		132	PRINTER
05	F									
06	F									
07	F									
08	F									
09	F									
10	F									
	F									
	F									

If MFCM, change record length to 80 and device names to MFCM1 and MFCM2

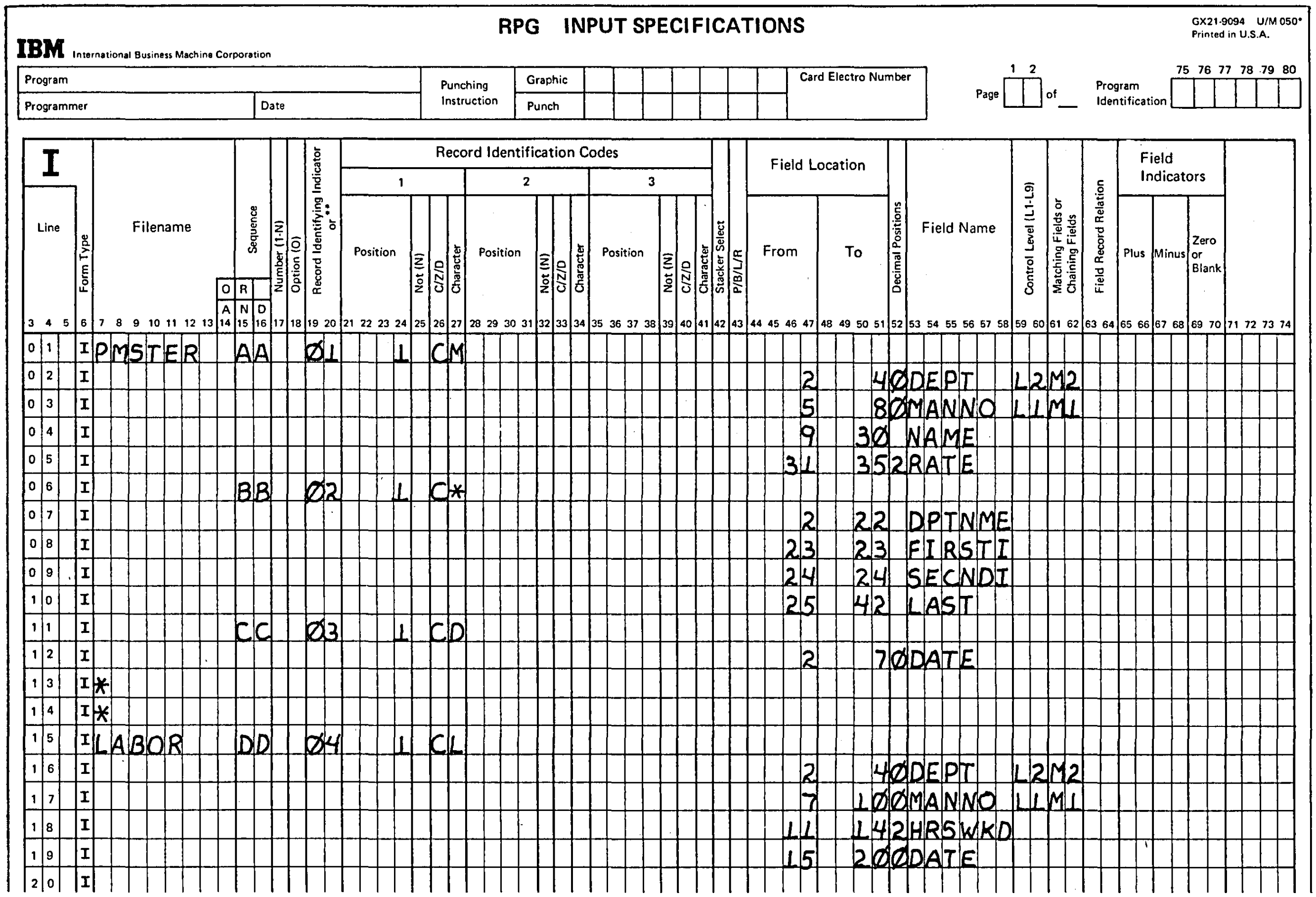
b. Disk files (sequential):



File Description Specification

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V/S/M/D	Block Length	Record Length	Device	Number of Extents
02	F	PMSTER	I	P		A	F	256	128	DISK	Ø1
03	F	LABOR	I	S	E	A	F	96	96	DISK	Ø1
04	F	REPORT	O				F	132	132	PRINTER	
05	F										
06	F										
07	F										
08	F										
09	F										
10	F										
	F										
	F										

2. Input Specifications



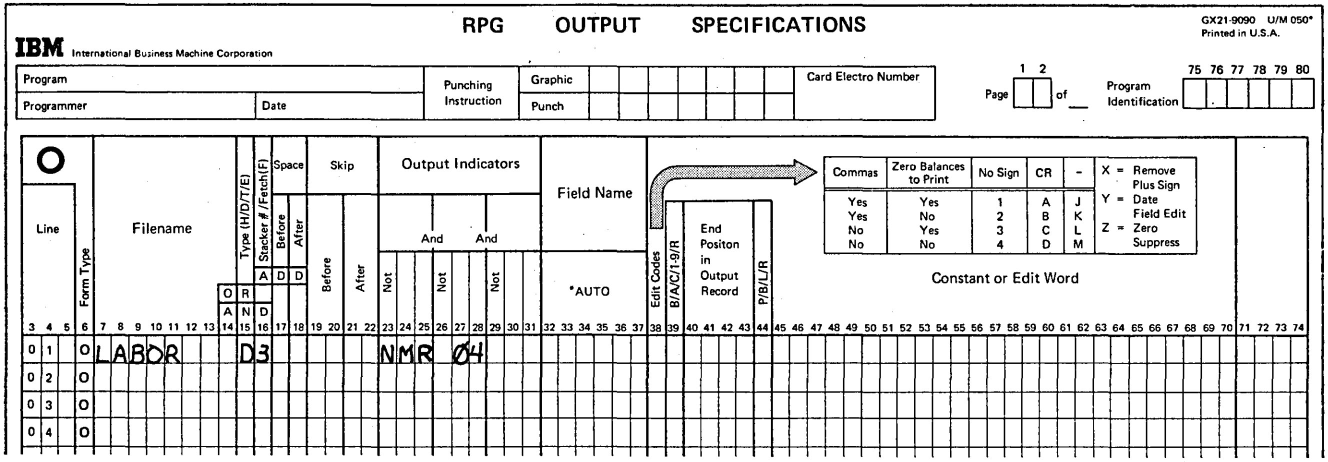
IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

```
01 IPMSTER  AA  Ø1   1 CM
02 I                                     2   4ØDEPT  L2M2
03 I                                     5   8ØMANNO L1M1
04 I                                     9  3Ø NAME
05 I                                    31  352RATE
06 I        BB  Ø2   1 C*
07 I                                     2  22 DPTNME
08 I                                    23  23 FIRSTI
09 I                                    24  24 SECNDI
10 I                                    25  42 LAST
11 I        CC  Ø3   1 CD
12 I                                     2   7ØDATE
13 I*
14 I*
15 ILABOR   DD  Ø4   1 CL
16 I                                     2   4ØDEPT  L2M2
17 I                                     7  1ØØMANNO L1M1
18 I                                    11  142HRSWKD
19 I                                    15  2ØØDATE
20 I
```

3. Output Specifications



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

```
01 OLABOR   D3      NMR Ø4
02 O
03 O
04 O
```

Chapter 7. Programmed Control of Input and Output

CHAPTER 7 DESCRIBES:

How to alter the order of processing input files using FORCE.

How to read one or more records per cycle from a demand file.

How to do repetitive or exception output during calculations.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Multifile processing.

Look-ahead.

End-of-file condition.

*PLACE.

RPG II program logic (Chapter 1).

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

RPG II FORCE operation code.

Using FORCE with the look-ahead feature.

Using the RPG II READ operation code to process demand files.

RPG II EXCPT operation code.

Note: You can use the review questions contained in *Review 7* at the end of this chapter to test your comprehension of each topic in the chapter. Questions are grouped according to the topic to which they apply. Answers follow the review questions.

INTRODUCTION

Normally, records are read, identified, selected for processing, and output according to the fixed logic of the RPG II object program. Sometimes, however, you need to have direct control over the input and output of your program. RPG II provides several operation codes which give you this control. By using the FORCE operation code in your own calculation routine, you can override normal RPG II multifile logic for selecting input records for processing. You can also do certain kinds of input and output during calculation time by using the READ and EXCPT operation codes.

Note: The CHAIN operation code also allows programmed control of input and output operations. The CHAIN operation is explained in *IBM System/3 RPG II Disk File Processing Programmer's Guide,* GC21-7566.

ALTERING THE ORDER OF PROCESSING FILES (FORCE OPERATION)

RPG II uses two methods to determine the order in which records are processed in a multifile program.

- If match fields are not specified for either file, all records in the primary file are processed, followed by those in the secondary files in the order defined on the File Description sheets.
- When match fields are assigned, the RPG II logic of matching records determines from which file the next record is to be processed.

The order of processing determined by RPG II logic is appropriate for most of your multifile programs. However, for certain programs, it may be necessary to have some of the records in the two files processed in an order other than that in which RPG II logic would select the records.

A record can be processed out of order only if you indicate to the program that the file containing that record is to be forced. To do this, you must code additional specifications to override normal RPG II multifile logic.

Regardless of how your files are organized, the following situations require that you alter the order of processing:

1. Match fields cannot be assigned to the files and you wish to:
   a. Alternate processing between two files.
   b. Process a primary file record followed by a particular number of secondary file records.
   c. Process a secondary file record only when it matches a primary file record.
2. Match fields are assigned to both input files. You wish to process one primary file record, followed by matching secondary file records, then the rest of the matching primary records.

To alter the order of processing, you must first determine which file is to be processed—when and under which conditions. Once you know the order, the next step is to determine, for a particular programming cycle, whether the RPG II logic will select the appropriate record or if you must force the processing of that record.

The first record to be processed in any program can only be selected by RPG II logic in the usual way. Thereafter, to alter the order of processing, you tell the program to force a record from a file which would not ordinarily be processed next. Once the forced record is processed, and providing another record is not forced, the RPG II logic selects the next record in the usual way. This is the record which would have ordinarily been processed if the other file had not been forced.

Specifying the Next File to Process

To process a record out of its normal sequence, you specify on the Calculation sheet the FORCE operation code and the name of the file which is to be forced in the *next* program cycle (Figure 7-1).

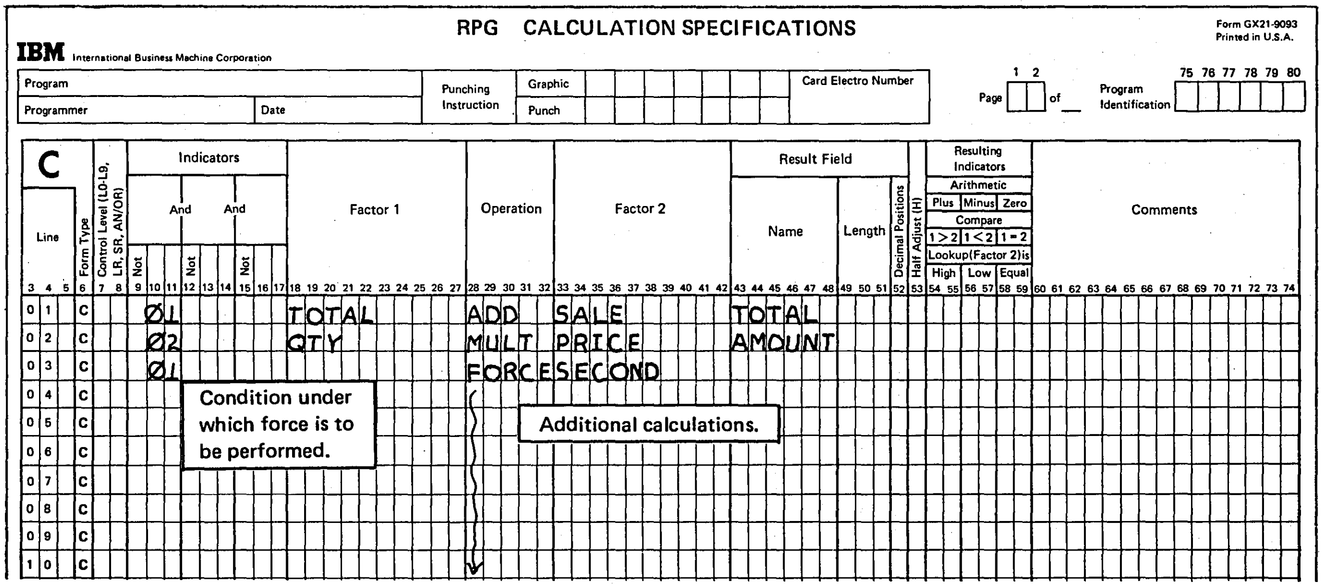
Assuming a record type 01 from the primary file is being processed, the calculation on line 01 is performed. The next detail-time calculation specification for record type 01 indicates that the secondary file (SECOND) is to be forced. The FORCE does not occur immediately, however. This specification only tells the program to remember that a record from the file SECOND is to be processed next. Any additional calculations and/or output for the record being processed are performed first to complete the present program cycle.

At the beginning of a normal program cycle, RPG II logic looks at the two records available in order to select the one to process during that cycle. However, if the record from the file which would not normally be selected is to be processed, this must be indicated to the program before the beginning of the cycle. If a file is to be forced, there is no need for RPG II logic to compare the records and make a selection. This is the reason that, if a file is to be forced, the FORCE must be indicated during the program cycle immediately before the cycle in which the FORCE is to occur.

Depending on your program, you may not have to force a record in every program cycle. For such situations, you must indicate when the FORCE is to be done by specifying conditioning indicators in columns 9-17 of the Calculation sheet (Figure 7-1). Whether the FORCE is to be performed in the next cycle or not may depend on any of several conditions:

- The type of file or record type being processed at the time.
- The number of records which have been processed.
- The result of a calculation performed.
- The contents of a field on the record being processed.
- The contents of a field on a record which has not been processed yet.

With these points in mind, let's discuss a job in which you can determine if the order of processing must be altered on the basis of the type of record being processed.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page □□ of __ | Program Identification

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C		Ø1	TOTAL	ADD	SALE	TOTAL					
02	C		Ø2	QTY	MULT	PRICE	AMOUNT					
03	C		Ø1		FORCE	SECOND						
04	C											
05	C											
06	C											
07	C											
08	C											
09	C											
10	C											

Condition under which force is to be performed.

Additional calculations.

Figure 7-1. Specifying the File to be Processed in the Next Program Cycle

Alternating Processing Between Two Files

The two files in Figure 7-2 each contain one record for every salesman in a company. The MASTER records are arranged in alphabetical order by salesman name; the MONTH file is arranged in ascending sequence by salesman number. Although there is no common match field, the records in the two files are, nevertheless, in a one-to-one correspondence. Salesmen numbers have been assigned such that they correspond to the order of the salesmen names. Thus, the first record in each file is for Baker (salesman #10); the second record in each file is for Costello (salesman #20), and so on.

The two files are to be processed to determine the amount of commission earned by each salesman. To do this, a salesman's commission rate from his MASTER record must be multiplied by the amount of his sales contained on his MONTH record (Figure 7-2). The calculated commission is then recorded in the salesman's MONTH record.

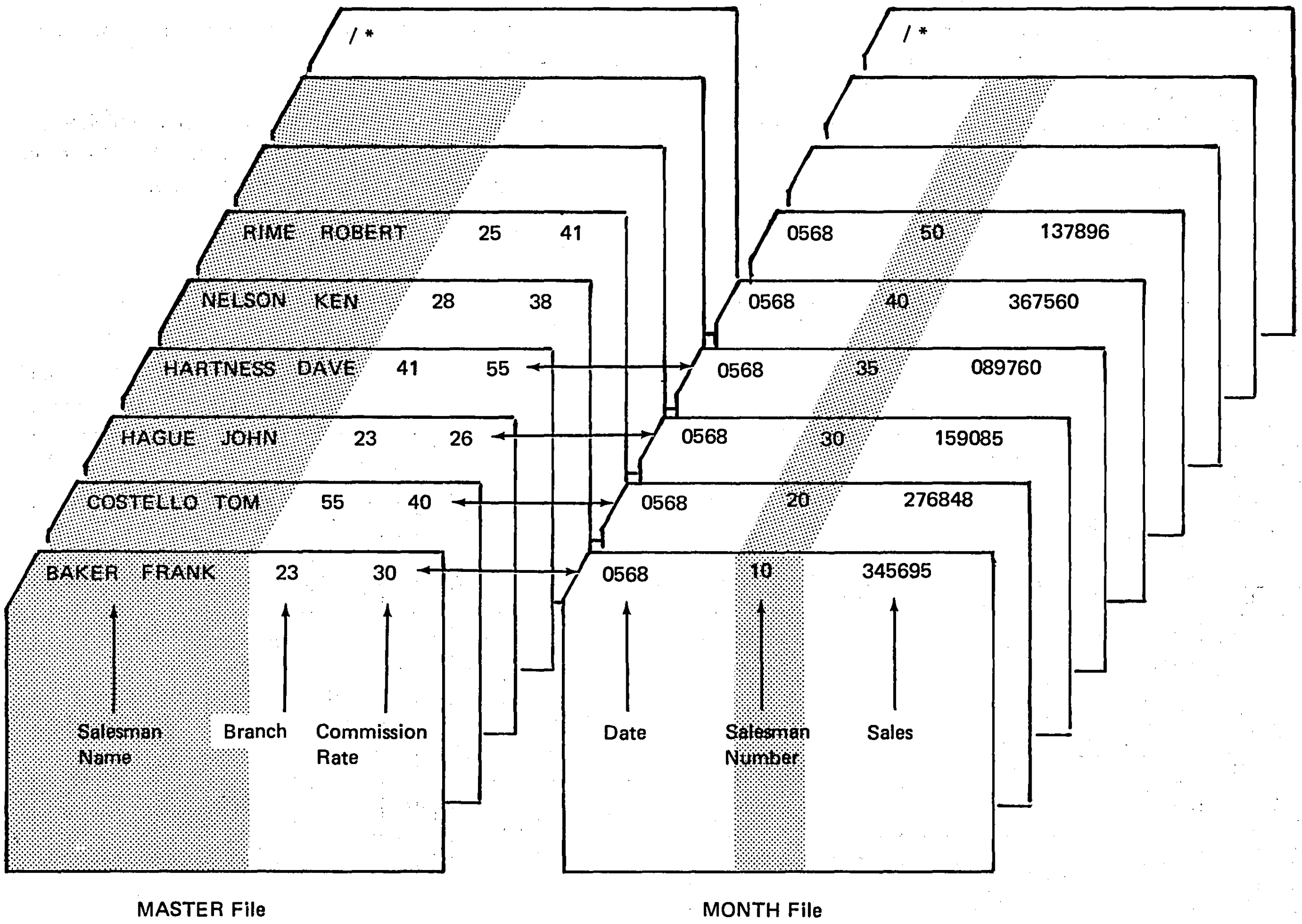
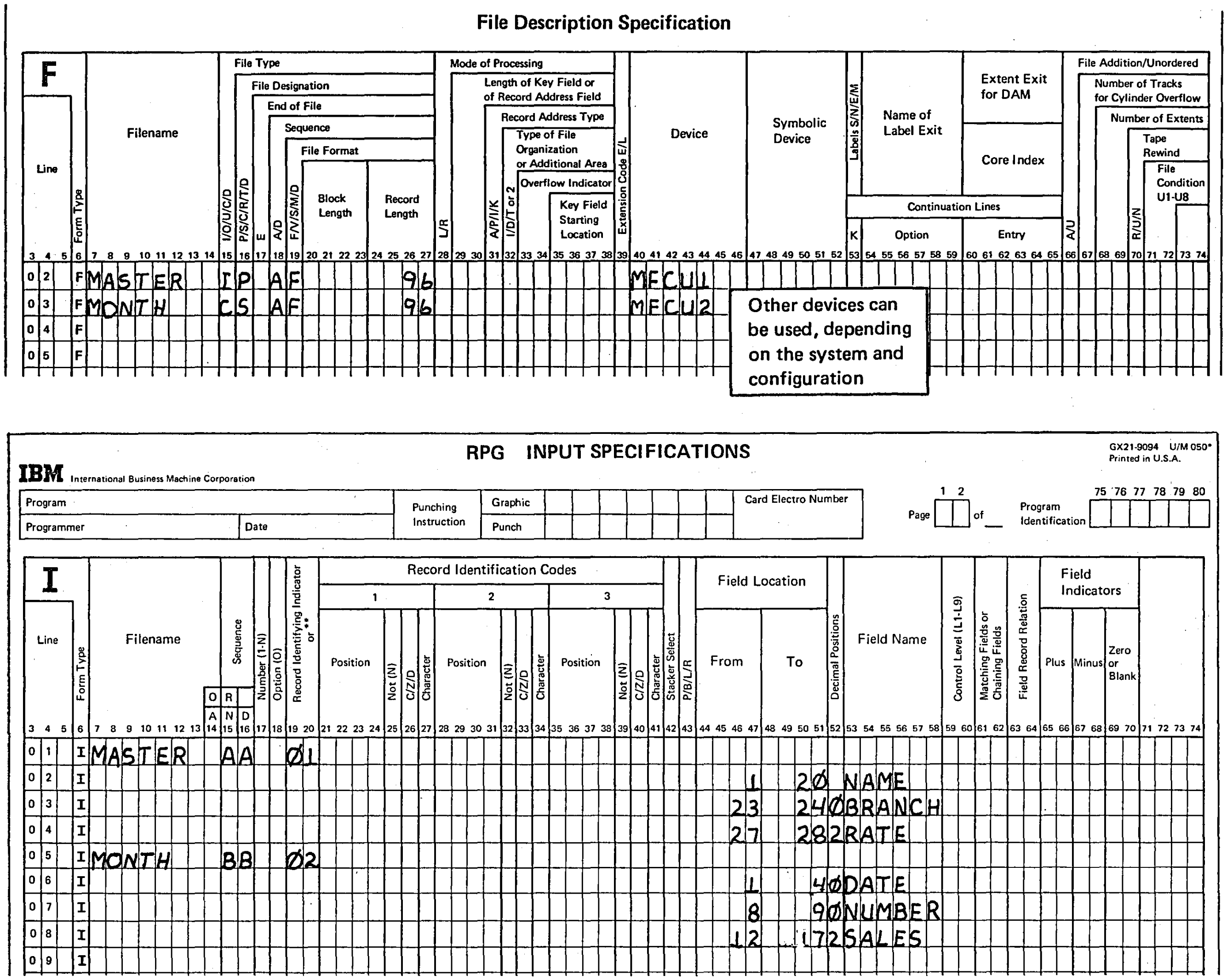


Figure 7-2. Files to be Alternately Processed

The two files are defined and the records described with the specifications shown in Figure 7-3. MASTER, the primary file, is assigned record identifying indicator 01. Indicator 02 is assigned to the secondary file, MONTH. Since MONTH is a card file to be used for both input and output, it is defined as a combined file. (MONTH could also be defined as an update file, on another device.)

This program must process the two files alternately; that is, every primary file record processed must be followed by a secondary file record. Likewise, every secondary file record processed must be followed by another primary file record. In this case, the MASTER record for a salesman is read to make the commission rate available. His MONTH record is then processed, to calculate the commission and record the amount in the MONTH record. Then, the next MASTER and the next MONTH record are processed in the same way for the second salesman.



File Description Specification

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	Device
02	F	MASTER	I	P		A	F		96	MFCU1
03	F	MONTH	C	S		A	F		96	MFCU2
04	F									
05	F									

Other devices can be used, depending on the system and configuration

IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Field Location From	Field Location To	Decimal Positions	Field Name
01	I	MASTER	AA	Ø1				
02	I				1	2Ø		NAME
03	I				23	24	Ø	BRANCH
04	I				27	28	2	RATE
05	I	MONTH	BB	Ø2				
06	I				1	4	Ø	DATE
07	I				8	9	Ø	NUMBER
08	I				12	17	2	SALES
09	I							

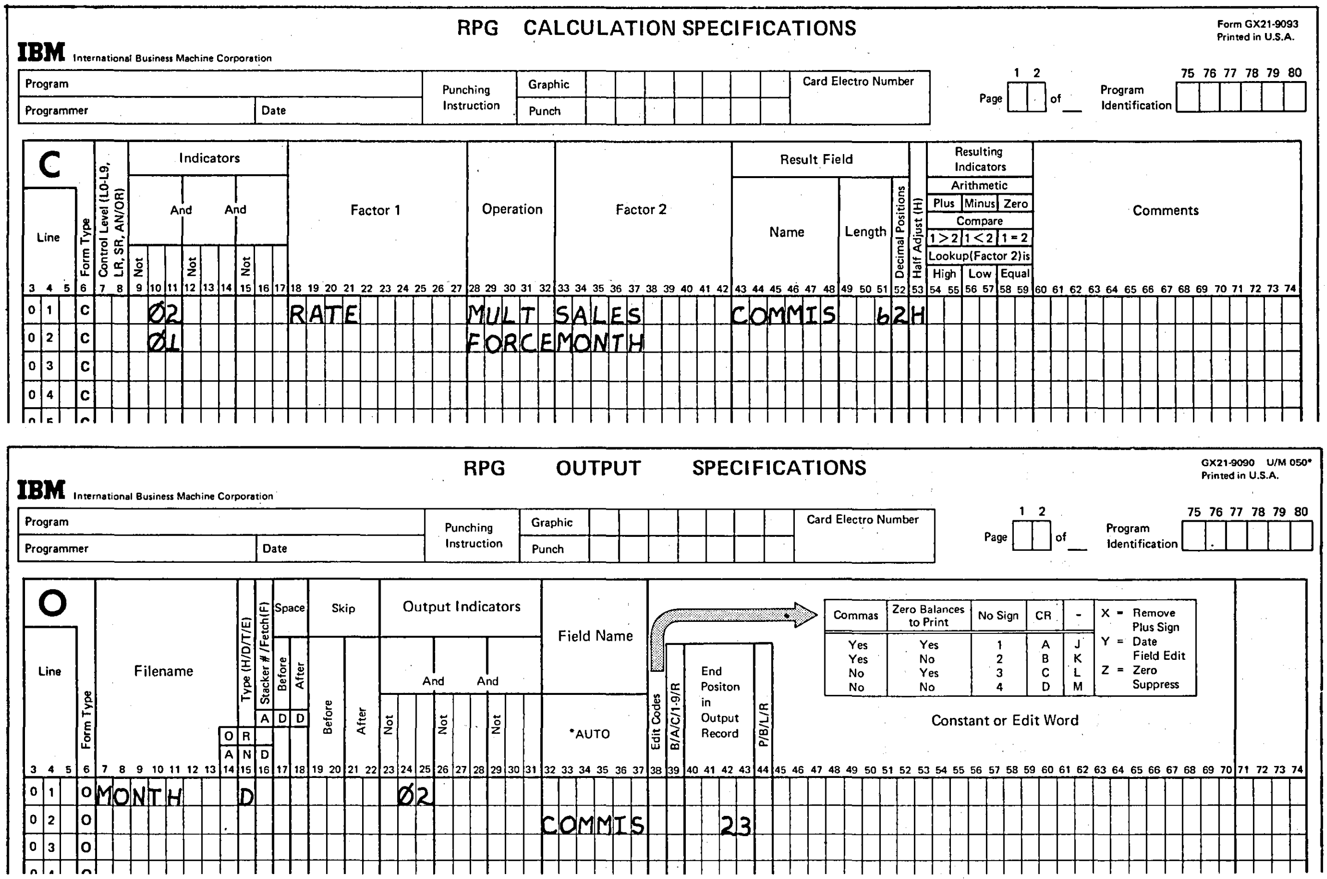
Figure 7-3. Specifications to Define and Describe Files to be Processed Alternately

Since no match fields are assigned, RPG II logic would normally process all records in the primary MASTER file before any of the secondary MONTH records. To alternate the files, you must tell the program to force a secondary file record every time a primary file record is being processed. When a MASTER record is being processed, record identifying indicator 01 is on. Thus, you should condition the FORCE operation to be performed only if 01 is on (Calculation sheet in Figure 7-4).

At the beginning of the program, RPG II logic automatically selects the first record, which is a primary file record. Since 01 is on, the FORCE operation is performed; that is, the program notes that a MONTH record is to be forced at the beginning of the next cycle.

During processing of the forced record (02 on), the commission is calculated and recorded in the MONTH record (Figure 7-4). Only the specifications for record type 02 are done. Therefore, the FORCE operation is not performed.

Since no FORCE was indicated in this cycle (02 on), RPG II logic again takes over to select a record, at the beginning of the next cycle. Thus, the second primary file MASTER record is processed (01 on). Processing of the files continues with the next MONTH record being forced and then another MASTER record being selected by RPG II logic until all records have been processed.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)
0 1	C	Ø2	RATE	MULT	SALES	COMMIS	6	2	H
0 2	C	Ø1		FORCE	MONTH				
0 3	C								
0 4	C								

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record
0 1	O	MONTH	D	Ø2		
0 2	O				COMMIS	23
0 3	O					

Figure 7-4. Conditioning Operations on Basis of Record Type

Forcing a Number of Records from a File

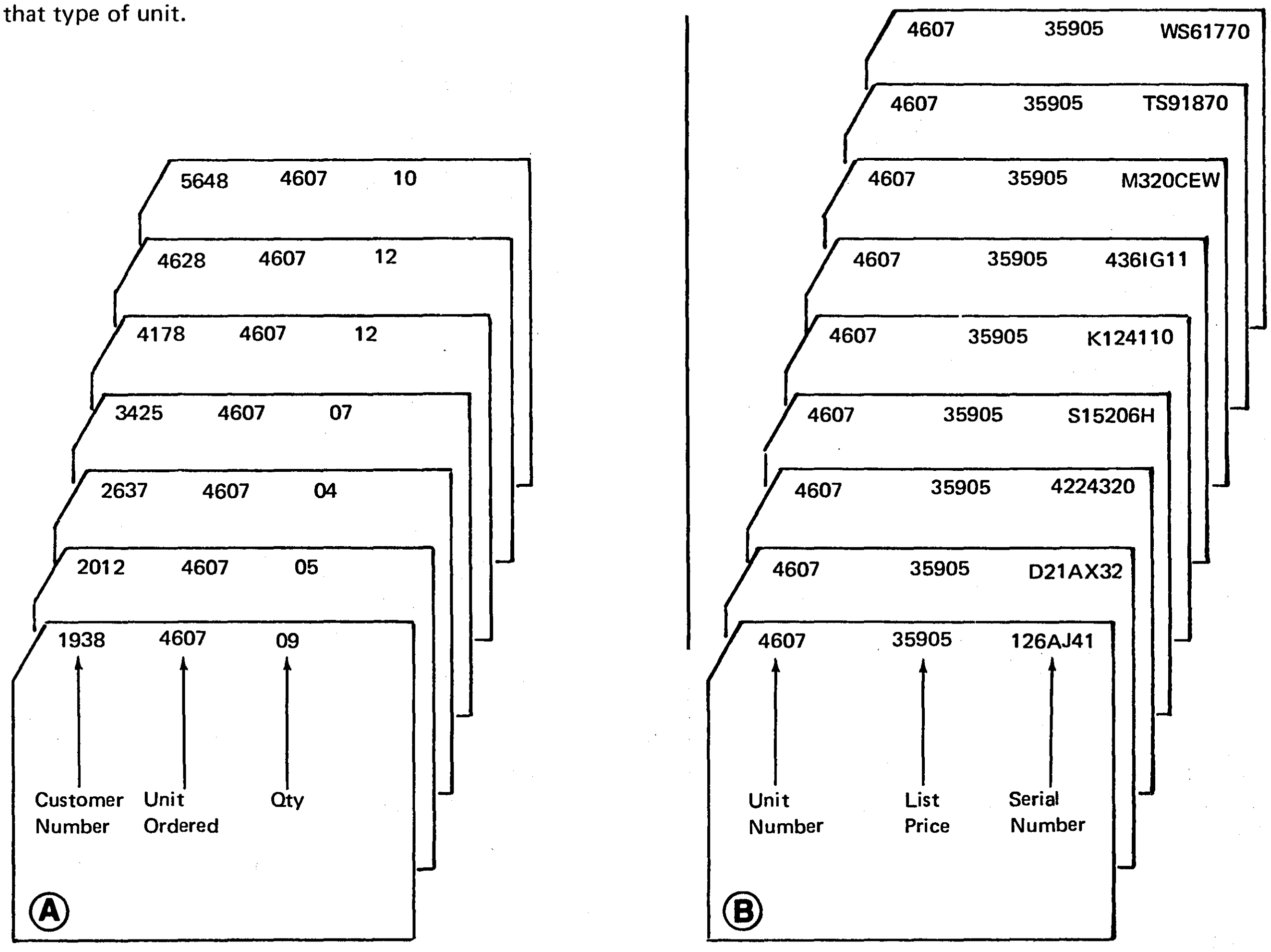
In the last example, the FORCE operation was performed only if a particular record identifying indicator was on. Now let's consider a case in which you condition the FORCE operation on the basis of whether a resulting indicator is on or off.

Suppose you have a number of customers who periodically order items to be delivered from a central warehouse. One record is kept for each unit in stock in the warehouse, and another record for each customer's order of a particular unit.

Orders are processed according to the type of unit ordered. Therefore, for a particular run, the primary file (ORDER) contains all order records for only one type of unit, and the secondary file (STOCK) contains all in-stock records for that type of unit.

For this run, the ORDER file (Figure 7-5, insert A) contains the week's order records for television sets, unit number 4607. The records show which customer placed the order, and the quantity of television sets wanted. The STOCK file consists of a separate record for each television set (unit 4607) in stock (Figure 7-5, insert B). Each record provides the unit number, list price, and serial number of the item.

There are two purposes for processing the files. First, you want an indication of which orders can be filled and which orders cannot be filled. Secondly, the STOCK file is to be kept up-to-date so it only contains as many records as there are television sets available.



ORDER file

STOCK file

Figure 7-5. Files for Processing Customer Orders

Note: System/3 will allocate two decimal places in the list price, although a decimal does not appear in the field. The list price in the stock file is logically represented as $359.05.

The program should produce a printed report showing which orders can be filled, and the amount each customer owes (Figure 7-6). Thus, you must determine the amount due for each item and the total amount for each order. A record from each file must be available before you can calculate the information.

Files for this program must be processed in a specific order. The quantity ordered (QTY) from an ORDER record must be available first. This quantity is used to determine how many STOCK records are to be processed. When enough STOCK records for an order have been processed, the next ORDER record is selected to repeat the process.

Looking at the two files, you can see that every record has a common field containing the unit number. It does no good to assign a match field to control processing order, because the unit number is always the same for every record. All records in the primary file would be processed before any secondary file records.

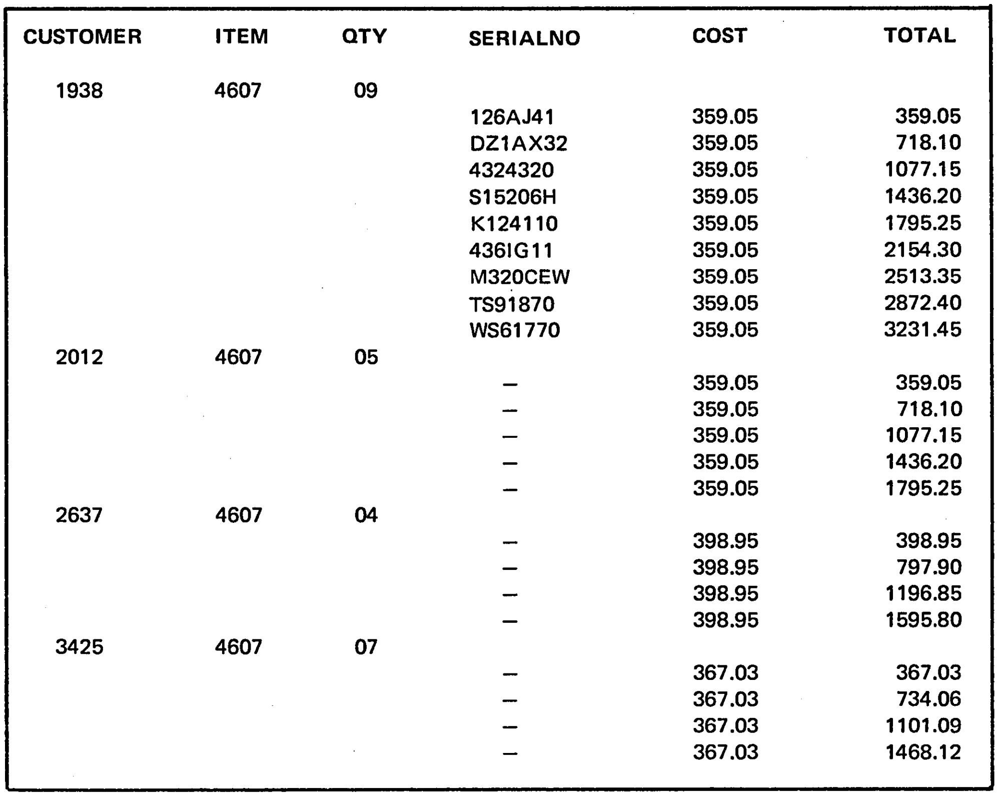
Remember, there is no way you can control selection of the first record to be processed in a program. RPG II logic always selects a primary file record first when match fields are not specified. Since an ORDER record must be available first for this program, the ORDER file should be designated as the primary file.

Controlling the Number of Times FORCE is Performed

After RPG II selects and processes an ORDER record, you must FORCE the processing of a number of STOCK records. The quantity ordered (QTY) from the ORDER record is used to control the number of times you force secondary file records. The quantity is stored in a field, called COUNT, to keep track of how many records are left to be forced for an order. Each time a STOCK record is forced, the number in COUNT is reduced by one. When COUNT reaches zero, enough STOCK records have been processed for that particular order. Then RPG II logic can again take over to process the next ORDER record (Figure 7-7).

The calculation specifications shown for this program (Figure 7-8) only determine if a record is to be forced in the next cycle.

Assume the first ORDER record (record type 01) has been selected, making the quantity ordered (QTY) available. The first calculation specification (line 01) for this record type stores the quantity in the COUNT field. Then the program determines if any STOCK records are to be processed for this order (line 05). If COUNT is greater than zero, indicator 27 turns on. With 27 on, line 06 is performed, indicating a STOCK record must be forced in the next program cycle.



```
CUSTOMER      ITEM      QTY     SERIALNO        COST          TOTAL

  1938        4607       09
                                126AJ41         359.05        359.05
                                DZ1AX32         359.05        718.10
                                4324320         359.05       1077.15
                                S15206H         359.05       1436.20
                                K124110         359.05       1795.25
                                4361G11         359.05       2154.30
                                M320CEW         359.05       2513.35
                                TS91870         359.05       2872.40
                                WS61770         359.05       3231.45
  2012        4607       05
                                   -            359.05        359.05
                                   -            359.05        718.10
                                   -            359.05       1077.15
                                   -            359.05       1436.20
                                   -            359.05       1795.25
  2637        4607       04
                                   -            398.95        398.95
                                   -            398.95        797.90
                                   -            398.95       1196.85
                                   -            398.95       1595.80
  3425        4607       07
                                   -            367.03        367.03
                                   -            367.03        734.06
                                   -            367.03       1101.09
                                   -            367.03       1468.12
```

Figure 7-6. Printed Report Showing Customer Orders Processed

At the beginning of the second cycle then, the first STOCK record (record type 02) is selected (by being forced). Line 03 is performed to reduce the COUNT by one for this record being processed. The COUNT is then compared to zero again (line 05) to determine if any more STOCK records are to be processed for this order. If COUNT is still greater than zero (27 set on again), line 06 is performed again, indicating another STOCK record is to be forced at the beginning of the next cycle.

During processing of the second STOCK record, COUNT is again reduced by one (line 03). Assuming COUNT is now at zero, the COMPARE operation on line 05 sets indicator 27 off. With 27 off, the FORCE operation on line 06 is not performed during this cycle. At the beginning of the next program cycle then, RPG II selects the next ORDER record from the primary file in the usual way.

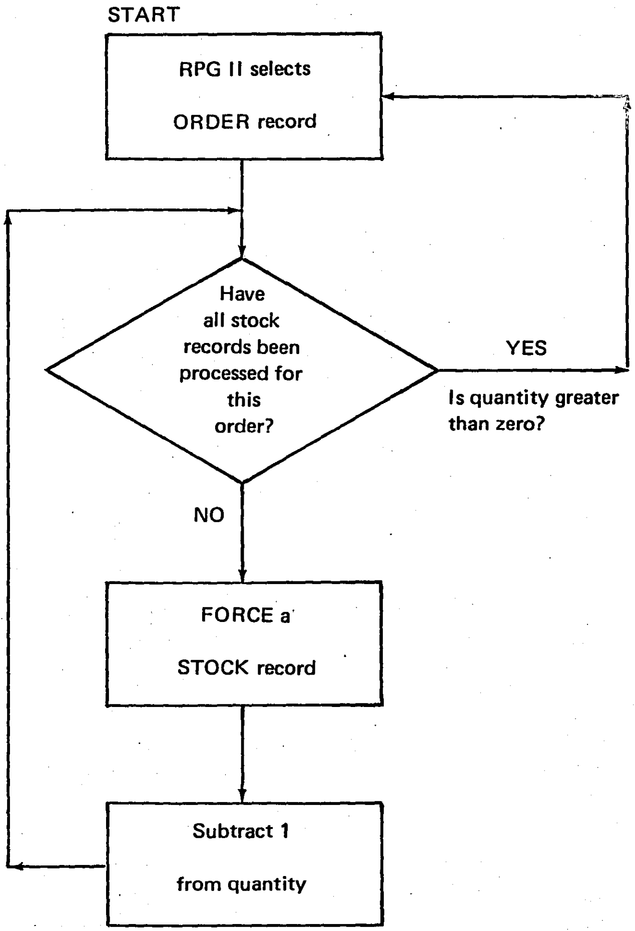
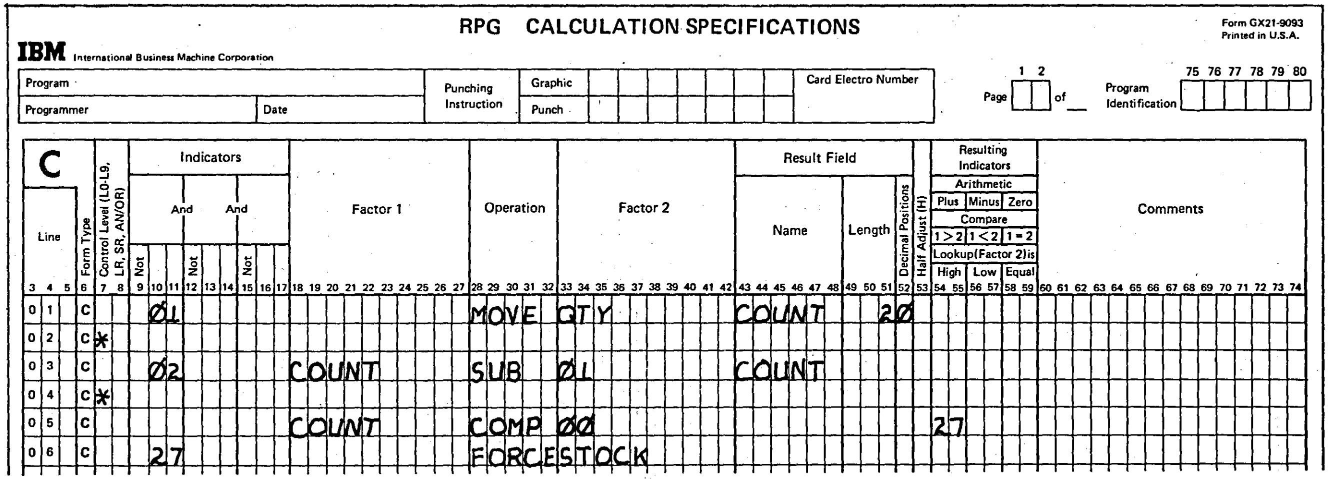

START
RPG II selects ORDER record
Have all stock records been processed for this order?
YES
Is quantity greater than zero?
NO
FORCE a STOCK record
Subtract 1 from quantity

Figure 7-7. Determining When Stock Records Must be Forced to Fill an Order



IBM International Business Machine Corporation

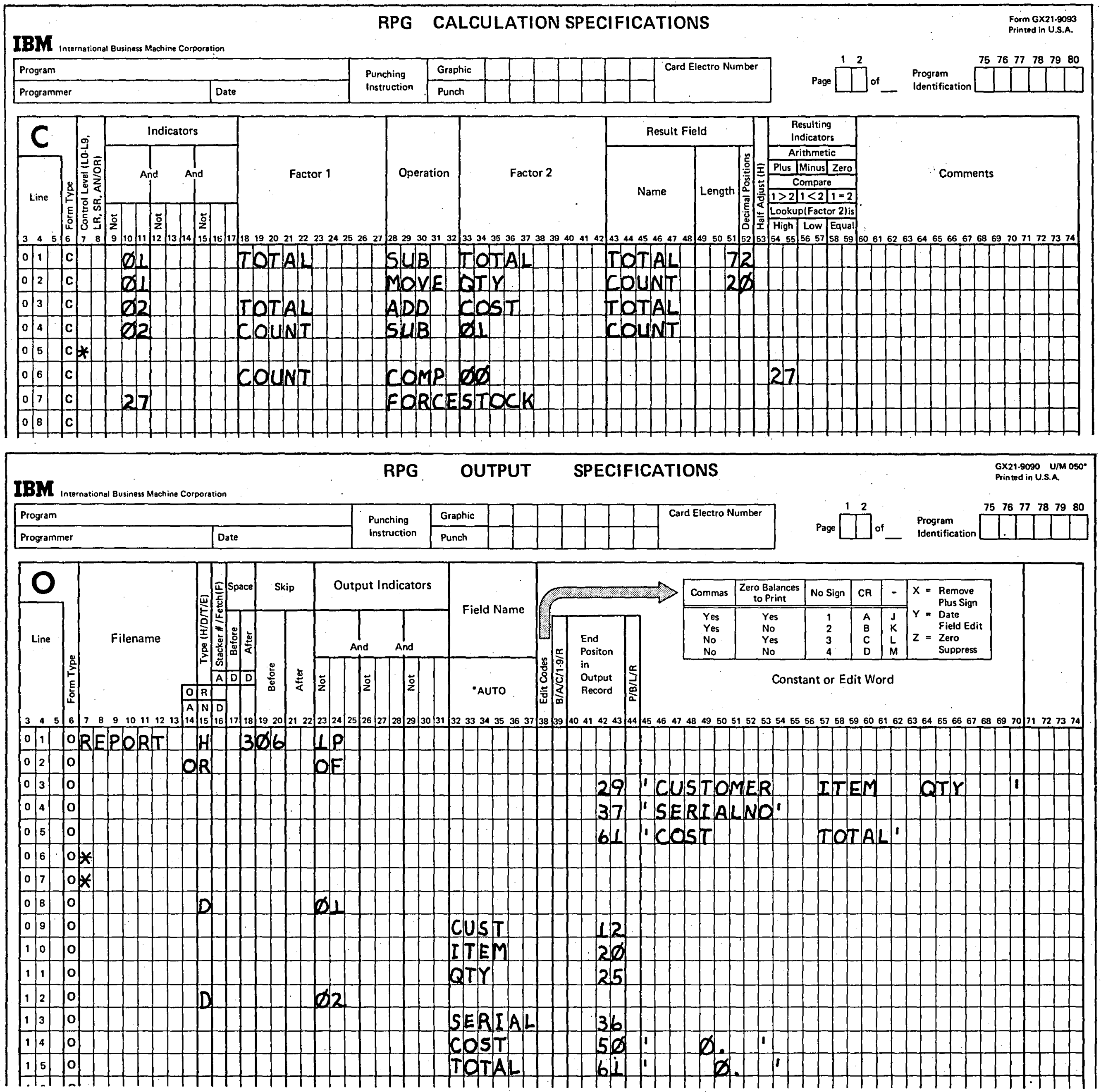
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Control Level	Not	Indicators	Not	And	Not	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	Plus / High	Minus / Low	Zero / Equal	Comments
01	C			Ø1						MOVE	QTY	COUNT	2	Ø					
02	C*																		
03	C			Ø2					COUNT	SUB	Ø1	COUNT							
04	C*																		
05	C								COUNT	COMP	ØØ					27			
06	C			27						FORCE	STOCK								

Figure 7-8. Controlling the Number of Times a File is Forced

At this point, add the specifications for calculating the amount due and for printing the report (Figure 7-9). An ORDER record is selected first, making the QTY available. Calculation lines 01, 02, and 06 in Figure 7-9 are performed for this record (record type 01). First, a TOTAL field, to be printed for each group of customers, is set to zero (line 01). Next, the quantity ordered is moved into the COUNT field (line 02). If COUNT is greater than zero, indicator 27 is turned on (line 06), and line 07 is performed; the program is instructed to force a STOCK record at the beginning of the next cycle. Before forcing, however, the output specifications (Figure 7-9, lines 08-11) are performed to print data from the ORDER record.



IBM International Business Machine Corporation
RPG CALCULATION SPECIFICATIONS
Form GX21-9093
Printed in U.S.A.

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Resulting Indicators Plus/High	Comments
01	C		Ø1	TOTAL	SUB	TOTAL	TOTAL	7	2		
02	C		Ø1		MOVE	QTY	COUNT	2	Ø		
03	C		Ø2	TOTAL	ADD	COST	TOTAL				
04	C		Ø2	COUNT	SUB	Ø1	COUNT				
05	C	*									
06	C			COUNT	COMP	ØØ				27	
07	C		27		FORCE	STOCK					
08	C										

IBM International Business Machine Corporation
RPG OUTPUT SPECIFICATIONS
GX21-9090 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Skip Before	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	REPORT	H	3	Ø6	LP			
02	O	OR				OF			
03	O							29	'CUSTOMER ITEM QTY '
04	O							37	'SERIALNO'
05	O							61	'COST TOTAL'
06	O*								
07	O*								
08	O		D			Ø1			
09	O						CUST	12	
10	O						ITEM	2Ø	
11	O						QTY	25	
12	O		D			Ø2			
13	O						SERIAL	36	
14	O						COST	5Ø	' Ø. '
15	O						TOTAL	61	' Ø. '

Figure 7-9. Specifications to Process Customer Orders

Following output, the next cycle begins with a forced STOCK record (record type 02) being processed. The cost is added to a TOTAL field (line 03) to accumulate the total amount due on the order.

Then, COUNT is reduced by one for the record being processed (line 04). Once again COUNT is compared to zero (line 06) to determine if line 07 should be performed; that is, to determine if another STOCK record is to be forced for the next cycle. The COST and TOTAL calculated for the STOCK record are then printed, by performing the output specifications on lines 12-15.

The record selected at the beginning of the next program cycle depends on whether a FORCE operation was indicated in the previous cycle. If calculation line 07 had been performed, another STOCK record would be processed (by being forced). If not, RPG II would select the next ORDER record from the primary file.

Controlling Processing After Reaching End of File

In a multifile program, end of file entries can be specified for any, all, or none of the input files. If an end of file entry is specified for only one file, processing stops after all records from that file have been processed. (Remember, however, if match fields are assigned to the files, the program continues to process the records which match the last record processed or which have no match fields.) If end of file entries are specified for all or for none of the input files, processing continued until all records in all files have been processed.

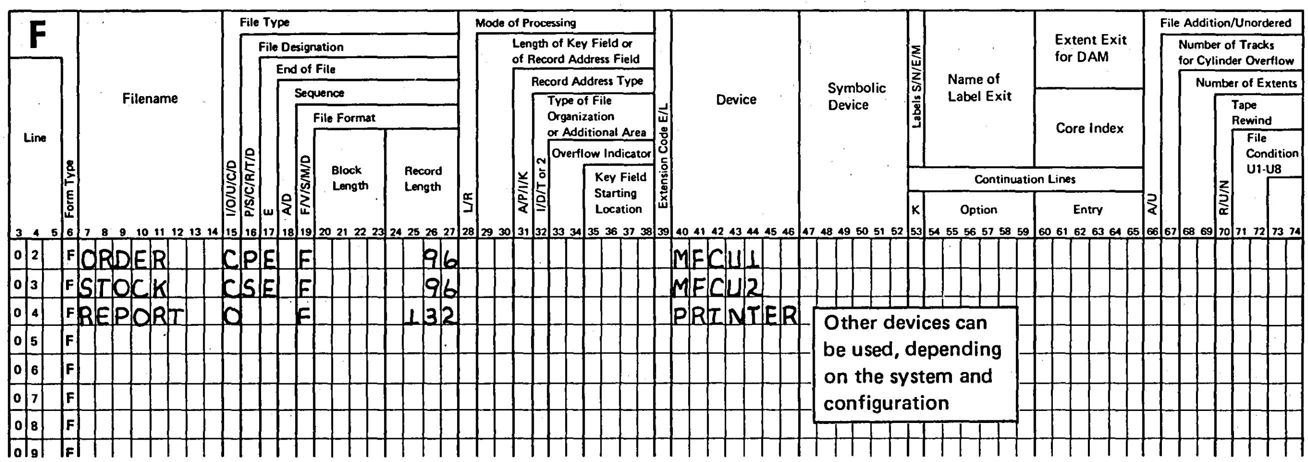
For this program, suppose ORDER and STOCK are card files, and processing must continue until both reach end of file. However, if one file runs out before the other, you don't want to perform the usual calculations and output for the remaining file. The remaining cards must be processed in another way; that is, by selecting them to a special stacker. If all orders are filled, any remaining STOCK cards should be selected to stacker 3. If there aren't enough items in stock to fill all orders, any remaining ORDER cards should be selected to stacker 2.

To continue the program until both files have been completely processed, specify end of file entries on the File Description sheet, either for both of or for neither of the two input files (Figure 7-10). By doing this, the LR indicator won't be set on to end the program until the last record of the last file has been processed.

With end of file specified for both files, let's consider what will happen when only one of the files reaches end of file (see Figure 7-9). First, suppose the STOCK file reaches end of file before all ORDER cards are processed. Since both files are not at end of file, processing will not stop. Instead, after processing the next ORDER card, the program will try to force the appropriate number of records from the STOCK file. With no more STOCK cards to force, another ORDER record will be selected. Once again, the program will try to force STOCK records for that order. The process continues until all primary file ORDER records have been processed. Furthermore, every time a new ORDER record is processed, it is printed on the report and the card goes into stacker 1, as if the order were being filled.

A similar problem arises if the ORDER file reaches end of file first. Suppose the last order has just been filled.

File Description Specification



Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	Device
02	F	ORDER	C	P	E		F		96	MFCU1
03	F	STOCK	C	S	E		F		96	MFCU2
04	F	REPORT	O				F		132	PRINTER
05	F									
06	F									
07	F									
08	F									
09	F									

Other devices can be used, depending on the system and configuration

Figure 7-10. Continuing Processing Until Both Files Reach End-of-File

After processing the last STOCK card for that order, no more STOCK cards are to be forced, so RPG II logic tries to select the next primary file ORDER record. In doing so, the /* card from the ORDER file is read. Since the program is to continue until both files are processed, RPG II logic automatically processes the remaining records in the other file (STOCK). As each remaining STOCK record is processed, the calculations and output for that record type are performed and the card goes into stacker 4, as if the record were being used to fill an order.

Although processing is to continue until both files reach end of file, you must have a way of determining when the first file runs out and which file it is. This is necessary so that normal calculations and output for the remaining cards can be bypassed and stacker selection specifications performed instead. Reading a /* card from a file will not necessarily indicate that end of file has been reached in all files. The LR indicator is turned on only when the /* card of the last file has been read. Therefore, a trailer card should be placed at the end of each file (Figure 7-11), before the /* card, to indicate when all cards of the file have been processed. When a trailer card is read, the record identifying indicator associated with that card turns on to indicate which file has been completely processed. For this program, the trailer card in the ORDER file will turn on indicator 03, while the trailer card in the STOCK file will turn on indicator 04.

Let's look at the completed calculation and output specifications for this program (Figure 7-12) to understand how the trailer cards are used to control processing.

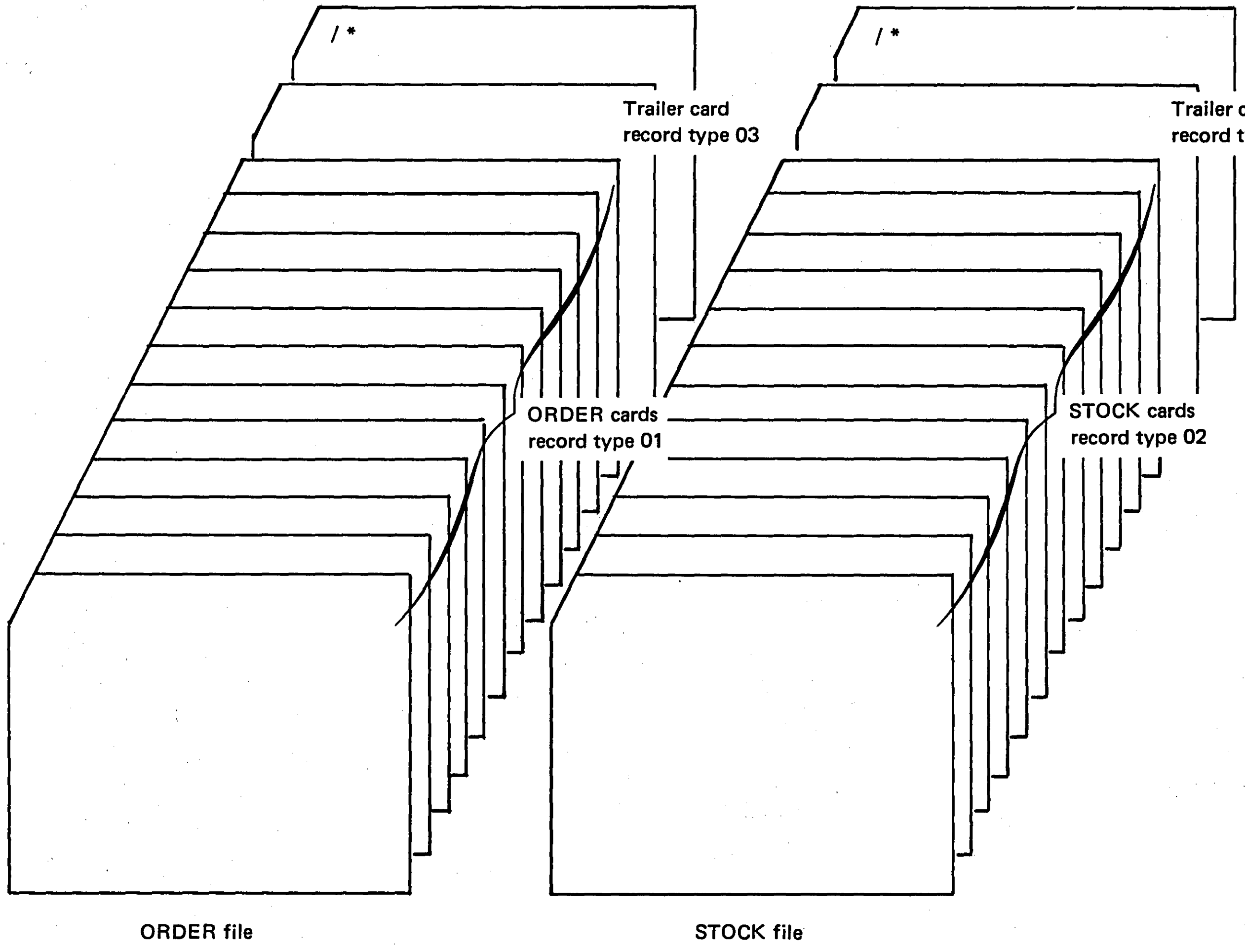


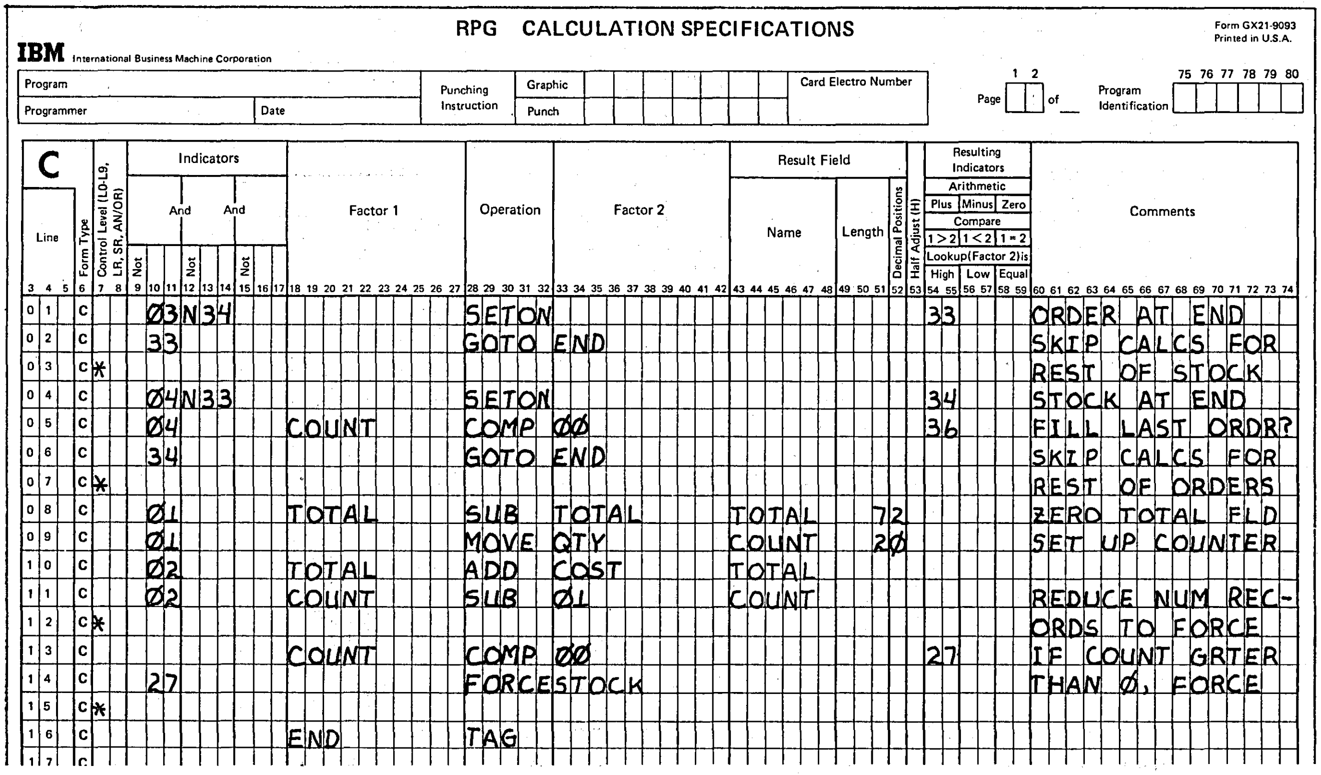
Figure 7-11. Use of Trailer Cards to Control End-of-File Processing

Assume the trailer card from the ORDER file (record type 03) is read, indicating all ORDER cards have been processed. When record identifying indicator 03 is set on, you know that any remaining STOCK cards are to be selected to stacker 3. However, indicator 03 cannot be used to condition the stacker selection. As soon as the trailer card has been processed and the first of the remaining STOCK cards read, indicator 03 is set off and record identifying indicator 02 (for the STOCK card) is set on. Therefore, while the trailer card is still being processed, you must set on an indicator which will stay on until the last STOCK card has been stacker selected. As calculation line 01 shows, record identifying indicator 03 is used to SETON indicator 33. (The use of N34 on line 01 will become clear as the program is explained.)

Now when a STOCK card is selected for processing, the program must determine if normal calculations and output should be performed for the record (record type 02). Since indicator 33 is on, normal calculations and output should be bypassed and the STOCK record stacker selected instead. To do this, all of the specifications for record type 02 could be conditioned to be performed only if indicator 33 is off (N33). However, to eliminate the extra coding and indicator testing, calculation line 02 instructs the program to skip the calculations when 33 is on, merely by branching to the end of calculations. Normal output for STOCK records is then conditioned so it isn't performed if indicator 33 is on (output lines 10-13). Output line 19 then instructs the program to stacker select the STOCK card if 33 is on.

Now, let's assume the STOCK file runs out before the ORDER file. When the trailer card from the STOCK file (record type 04) is read, you know that the remaining ORDER cards must be selected to stacker 2. For the same reason as previously explained, the record identifying indicator of the trailer card cannot be used to condition the stacker selection. Therefore, record identifying indicator 04 is used to SETON indicator 34 (Figure 7-12, calculation line 04).

In addition, while the trailer card is being processed, you must determine if the last order was completely filled. If there had been enough STOCK records to fill the order, the quantity in the COUNT field would have been reduced to zero. Therefore, when record identifying indicator 04 is on, COUNT field should be checked (calculation line 05). If COUNT is greater than zero, indicator 36 is set on. With 36 on, a message will be printed indicating how many items of the last order could not be filled (output lines 14-17).



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators (Not)	Indicators And (Not)	Indicators And (Not)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
3 4 5	6	7 8	9 10 11	12 13 14	15 16 17	18–27	28–32	33–42	43–48	49 50 51	52	53	54 55	56 57	58 59	60–74
01	C		Ø3	N34			SETON						33			ORDER AT END
02	C		33				GOTO	END								SKIP CALCS FOR
03	C	*														REST OF STOCK
04	C		Ø4	N33			SETON						34			STOCK AT END
05	C		Ø4			COUNT	COMP	ØØ					36			FILL LAST ORDR?
06	C		34				GOTO	END								SKIP CALCS FOR
07	C	*														REST OF ORDERS
08	C		Ø1			TOTAL	SUB	TOTAL	TOTAL	7	2					ZERO TOTAL FLD
09	C		Ø1				MOVE	QTY	COUNT	2	Ø					SET UP COUNTER
10	C		Ø2			TOTAL	ADD	COST	TOTAL							
11	C		Ø2			COUNT	SUB	Ø1	COUNT							REDUCE NUM REC-
12	C	*														ORDS TO FORCE
13	C					COUNT	COMP	ØØ					27			IF COUNT GRTER
14	C		27				FORCE	STOCK								THAN Ø, FORCE
15	C	*														
16	C					END	TAG									
17	C															

Figure 7-12 (Part 1 of 2). Controlling Operations at End-of-File

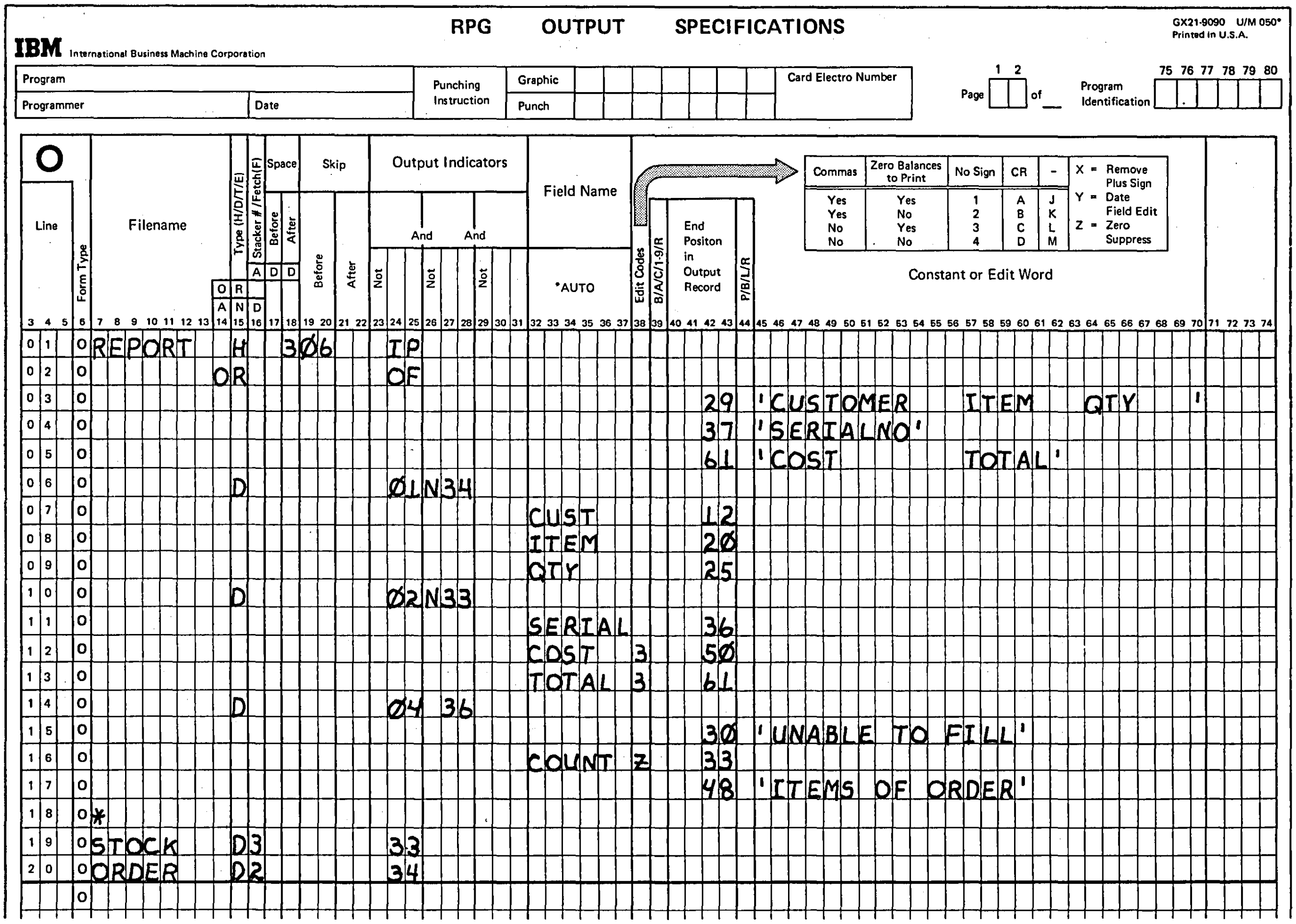
At this point, the first of the remaining ORDER cards is read. Since indicator 34 is still on, all calculations for the ORDER card are bypassed (see calculation line 06). The normal output for the ORDER record is to be performed only if indicator 34 is off. Thus, the printer output for record type 01 (output lines 06-09) is skipped and the record is selected into stacker 2 instead (output line 20).

As you have seen, the trailer cards from each file are used to set on particular indicators. When the ORDER file trailer card (record type 03) is read, indicator 33 is set on. Likewise, indicator 34 is set on when the STOCK file trailer card (record type 04) is read. The indicator which is set on (33 or 34) is used to prevent certain operations from being performed and to cause remaining records in the other file to be stacker selected.

Of course, reading a trailer card does not mean there are always cards remaining in the other file. For instance, assume all ORDER cards have been processed, but not all STOCK cards. When the ORDER file trailer card is read, it signals that the remaining STOCK cards should be stacker selected. As the STOCK cards are stacker selected, one-by-one, eventually you reach the end of STOCK by reading the trailer card from the STOCK file. At this point, both files have reached end of file, the LR indicator is automatically set on, and the program is ended.

When the ORDER file trailer card is read (03 set on), then you want to know if the STOCK file has been completely processed. If it has, the STOCK file trailer card (record type 04) has already been read, and the indicator set on by 04 (indicator 34) would still be on. With 34 on, there's no point in having record type 03 set on indicator 33. Likewise, if indicator 33 is on (ORDER at end of file) when the STOCK file trailer card (record type 04) is read, there's no point in having record type 04 set on indicator 34. For this reason, the calculations on lines 01 and 04 of Figure 7-12 have been conditioned by N34 and N33, respectively.

As you know, the setting of an assigned indicator can determine whether a FORCE is to be performed or not in the



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

```
01 OREPORT  H 3 Ø6   1P
02 O       OR        OF
03 O                                   29 'CUSTOMER   ITEM   QTY   '
04 O                                   37 'SERIALNO'
05 O                                   61 'COST       TOTAL'
06 O        D        Ø1N34
07 O                         CUST      12
08 O                         ITEM      2Ø
09 O                         QTY       25
10 O        D        Ø2N33
11 O                         SERIAL    36
12 O                         COST  3   5Ø
13 O                         TOTAL 3   61
14 O        D        Ø4 36
15 O                                   3Ø 'UNABLE TO FILL'
16 O                         COUNT Z   33
17 O                                   48 'ITEMS OF ORDER'
18 O*
19 OSTOCK   D3       33
20 OORDER   D2       34
   O
```

Figure 7-12 (Part 2 of 2). Controlling Operations at End-of-File following program cycle. Either a record type, data on a record, or the result of calculations performed during processing of a record can set the particular conditioning indicator on or off.

In the examples presented thus far, something about a record which was currently being processed determined whether the conditioning indicator was set on and whether an unprocessed record was forced. In the last example, the QTY field on the ORDER card determined how many STOCK records were to be forced.

Look-Ahead to Determine Whether a File is to be Forced

For some programs, you cannot determine whether a record is to be forced in the next cycle until you know something about the records which have not been processed yet. Thus, you must look ahead in one or both files at the next record which is not yet available for processing. In looking ahead, you may be checking to determine what record type is next, to see what data is on the next record, or to determine if the next record has the same match field as the record being processed. What you find in looking ahead can determine which file is to be processed next and, also, whether the file must be forced or not.

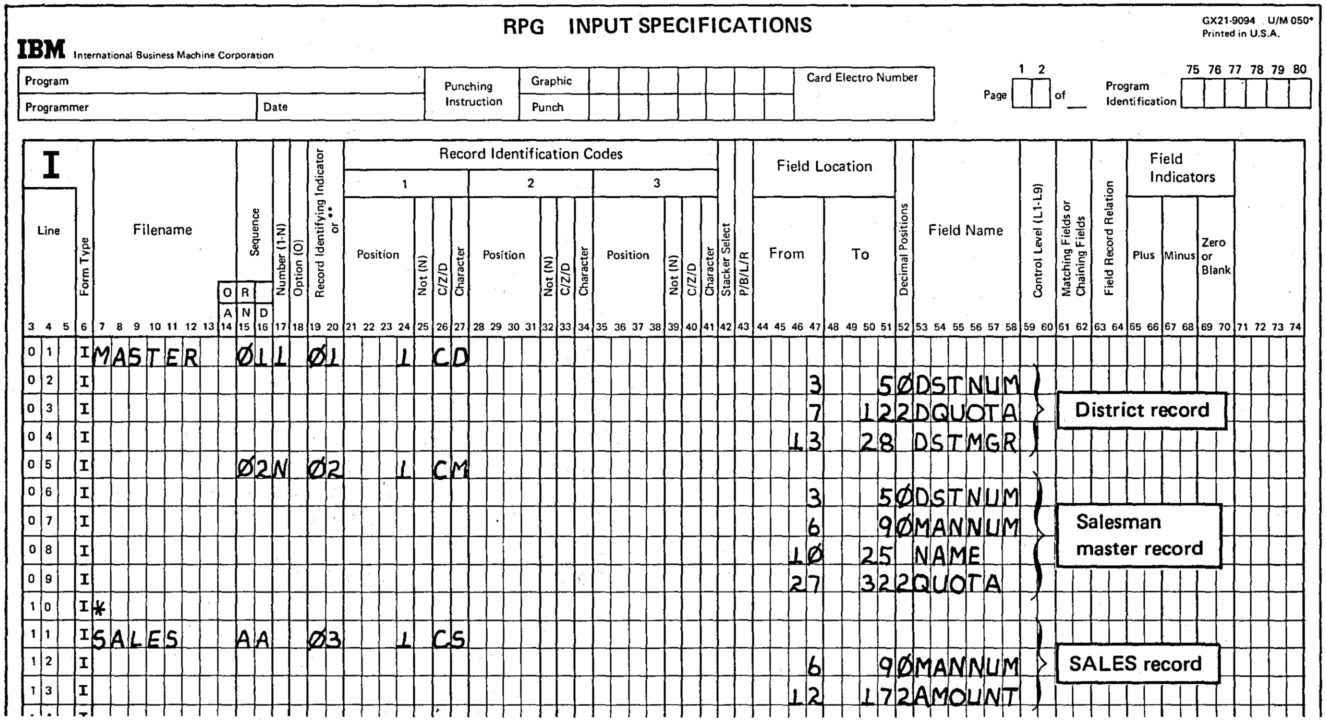
Before considering the use of FORCE with look-ahead, however, you should evaluate your system design. If at all possible, you should organize your files in such a way that the normal RPG II logic can determine the appropriate order of file processing. In this way, you do not have to code additional specifications to control the order. Of course, from time to time you may have jobs in which you must use FORCE and look-ahead.

Doing Matching Records Without Match Fields

If two files are organized such that the same match fields cannot be assigned to the two files, you can still process the matching records together by using look-ahead fields and the FORCE operation. (This cannot be done, however, if the look-ahead file is defined as a combined or update file.) Look-ahead can be used to determine if certain fields (not assigned as match fields) on an unprocessed record match those on the record being processed. If they match, FORCE is performed to cause the matching unprocessed record to be selected next.

As an example, assume a report is to be prepared showing the amount of each salesman's sales and his quota. The report should also compare the total of district sales with the district quota.

The two files available for this program are described in Figure 7-13. The primary file (MASTER) contains a district record (record type 01), followed by all salesmen's MASTER



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator (19-20)	Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)
01	I	MASTER	Ø1	1		Ø1	1		C	D				
02	I										3	5	Ø	DSTNUM
03	I										7	12	2	DQUOTA
04	I										13	28		DSTMGR
05	I		Ø2	N		Ø2	1		C	M				
06	I										3	5	Ø	DSTNUM
07	I										6	9	Ø	MANNUM
08	I										1Ø	25		NAME
09	I										27	32	2	QUOTA
10	I	*												
11	I	SALES	AA			Ø3	1		C	S				
12	I										6	9	Ø	MANNUM
13	I										12	17	2	AMOUNT

District record

Salesman master record

SALES record

Figure 7-13. Describing Files to be Matched Without Match Fields

records (type 02) associated with the district. These are in turn followed by the next district record (01) and its related salesmen MASTER records (02) and so on. Although the records are grouped by district, all salesman MASTER records in the file are still in ascending sequence by salesman number. The secondary file (SALES) contains only one record type (03), a record for each individual sale. The SALES file is also in ascending sequence by salesman number. While the MASTER file contains a record for every salesman (assume there are no MASTER records missing), the SALES file may contain only one, several, or even no records for a particular salesman.

To produce the report, the records should be processed in the following order:

1. District record (record type 01).
2. Salesman MASTER record (record type 02).
3. All SALES records for that salesman (record type 03).
4. Next salesman MASTER record.
5. SALES records for that salesman.
6. Next district record (after all salesman MASTER records associated with the first district have been processed).

There is no common field on all three record types which can be assigned as a match field to cause the records to be processed in this order. Totals are to be accumulated by salesman number; but the MANNUM field is contained only on the salesman MASTER and SALES records. The district records do not contain this information.

If the MANNUM fields are assigned as M1 match fields for only two of the record types, the records will be processed in an incorrect order. Following the last salesman MASTER record (02 record type) for a particular district, the next record in the same file is another district card. Since district records have no M1 match field entry assigned (no MANNUM field), the district record is processed immediately before the SALES records for the last salesman.

Although this program cannot be done using match fields, you can match the records yourself by using the look-ahead capability to compare fields (Figure 7-14). The object is to match a salesman's SALES record with this salesman MASTER record. Thus, the MANNUM field on a SALES record is defined as a look-ahead field. While processing a salesman MASTER record, you then look ahead (in calculations) at the unprocessed SALES record to determine if the salesman number is the same as on the record being processed. If they match, the FORCE operation code is used to process the SALES record as if RPG II logic were performing a matching records job. You then continue to force the rest of the SALES records which contain the same salesman number. For each record, look-ahead is used to check the MANNUM field to determine if the SALES record should be forced. When look-ahead indicates that the next SALES record is for a different salesman number, the SALES record is not forced. Instead, RPG II takes over to select the next primary file record which is either another salesman MASTER record or the next district record.

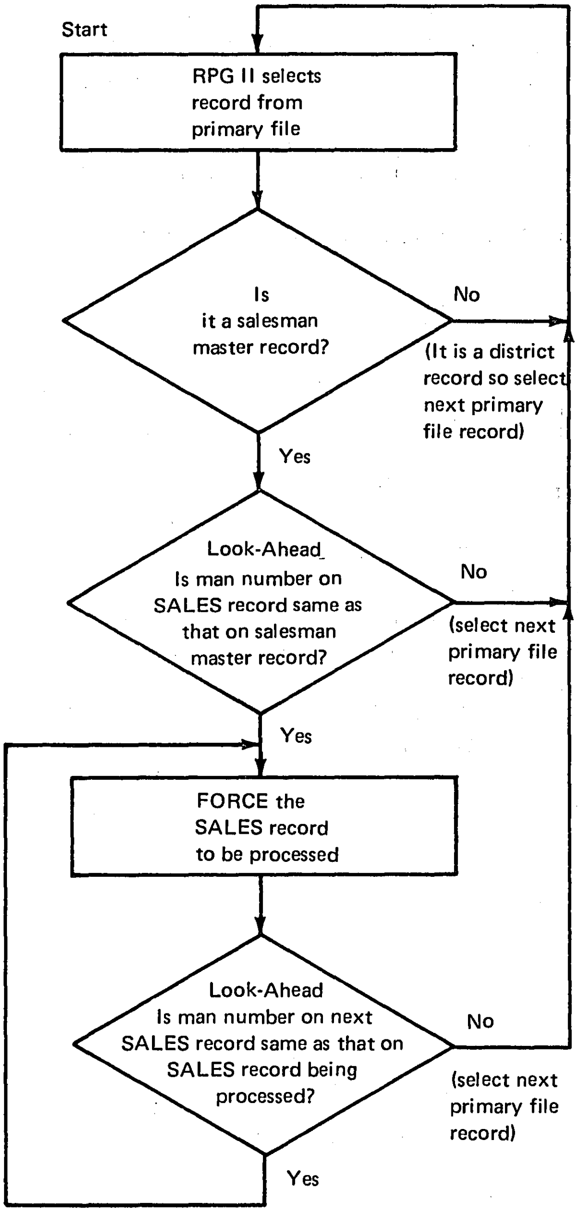
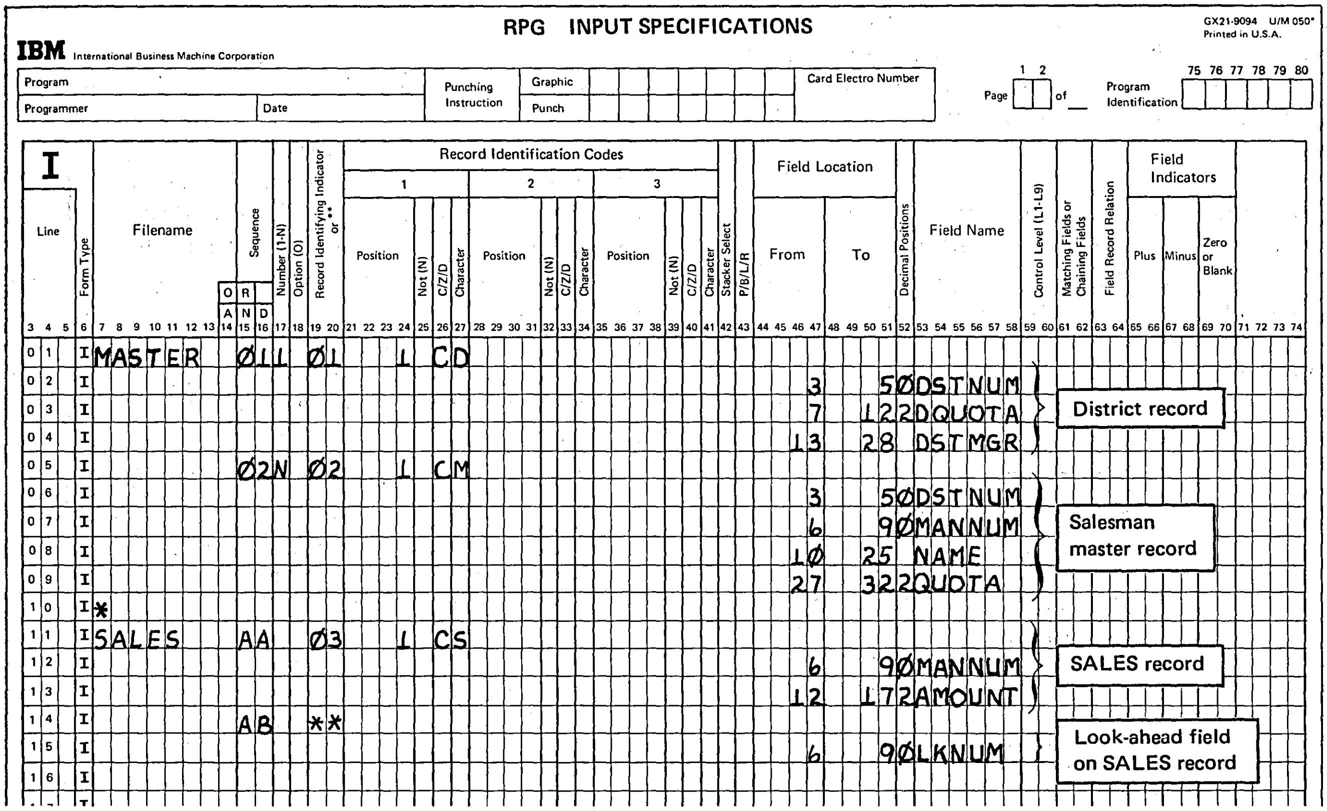

Start
RPG II selects record from primary file
Is it a salesman master record?
No
(It is a district record so select next primary file record)
Yes
Look-Ahead
Is man number on SALES record same as that on salesman master record?
No
(select next primary file record)
Yes
FORCE the SALES record to be processed
Look-Ahead
Is man number on next SALES record same as that on SALES record being processed?
No
(select next primary file record)
Yes

Figure 7-14. Using Look Ahead to Determine if Records Match

Now that you understand the steps involved in this program, look at the specifications in Figure 7-15. The Input sheet describes the records in each file and defines the look-ahead field for the SALES records. Only the calculations necessary to determine which record is to be processed next are shown. To actually prepare the report, you would need file description specifications to define the files and additional calculation and output specifications to accumulate totals and print the data.



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

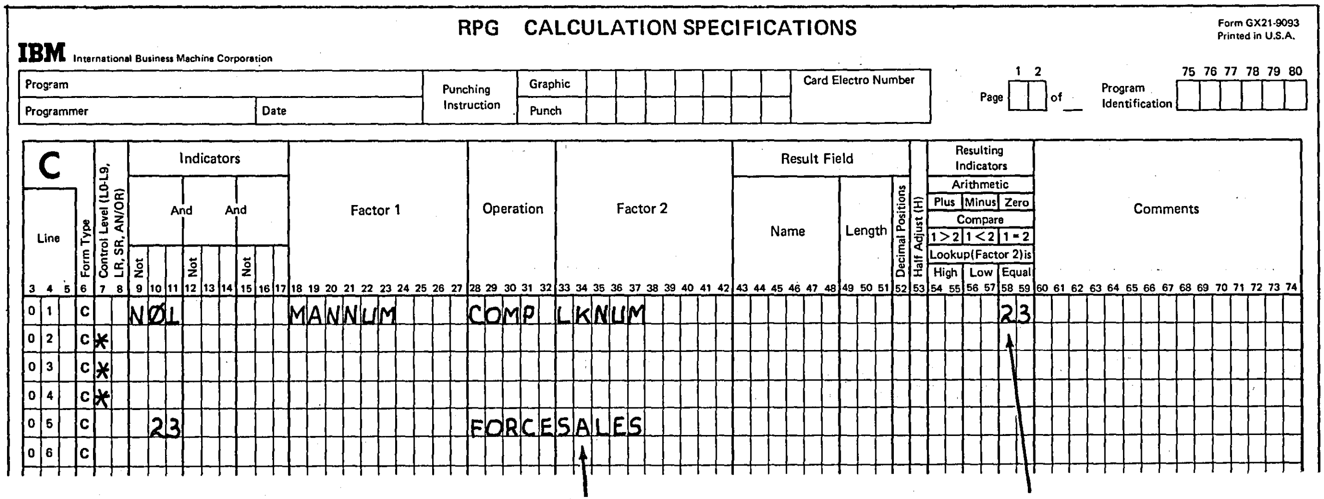
Line	Form Type	Filename	Sequence	Number (1-N)	Record Identifying Indicator or **	Position	C/Z/D	Character	From	To	Decimal Positions	Field Name
01	I	MASTER	Ø1	1	Ø1	1	C	D				
02	I								3	5	Ø	DSTNUM
03	I								7	12	2	DQUOTA
04	I								13	28		DSTMGR
05	I		Ø2	N	Ø2	1	C	M				
06	I								3	5	Ø	DSTNUM
07	I								6	9	Ø	MANNUM
08	I								1Ø	25		NAME
09	I								27	32	2	QUOTA
10	I*											
11	I	SALES	AA		Ø3	1	C	S				
12	I								6	9	Ø	MANNUM
13	I								12	17	2	AMOUNT
14	I		AB		**							
15	I								6	9	Ø	LKNUM
16	I											

District record

Salesman master record

SALES record

Look-ahead field on SALES record



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093 Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Resulting Indicators: Equal
01	C	NØ1	MANNUM	COMP	LKNUM	23
02	C*					
03	C*					
04	C*					
05	C	23		FORCE	SALES	
06	C					

If salesman numbers are the same (23 on), force the next SALES record.

If salesman master record (02) or SALES record (03) being processed, check salesman number on the next SALES record

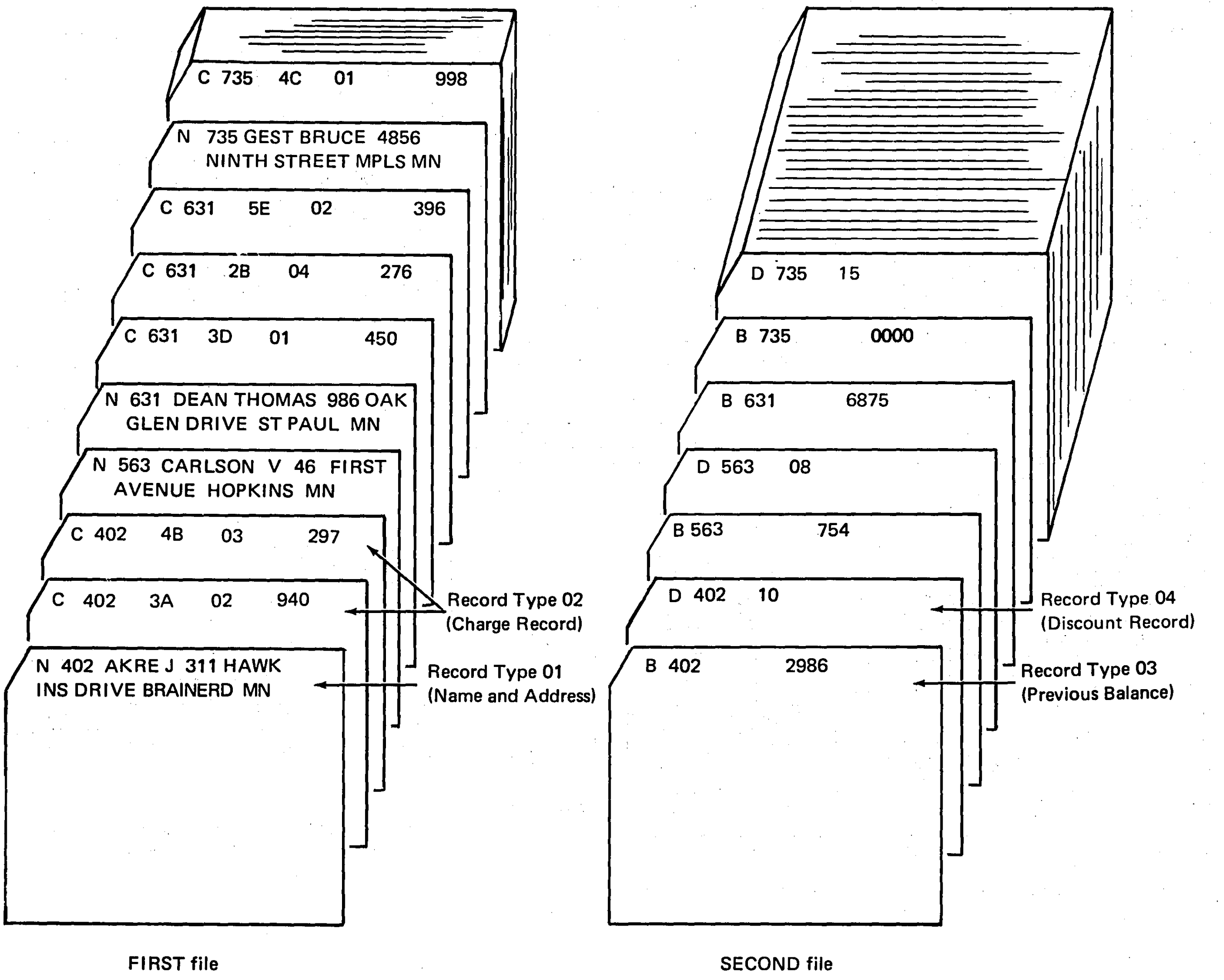
Figure 7-15. Using Look Ahead to Match Records

Processing a Primary, Then Matching Secondary Records Before Matching Primary Records

Another reason for looking ahead to determine which record to process next is when you want to process the files in an order other than the usual order of matching record logic. In such cases, all files must have match fields. The match fields on the next records available for processing are looked at to determine which record will ordinarily be selected and, thus, if it is necessary to force a record instead.

To illustrate altering the order of matching record logic, let's consider a sample billing program in which the records are to be processed in a particular order. However, note that the example is presented only to show you why and how look-ahead fields are used to determine whether a particular file should be processed. Actually, the desired results could also be obtained by processing the records in a different order using the normal RPG II logic. We are not necessarily suggesting that your files be organized in the same way for your billing purposes. In fact, it is possible that the files for the sample program could be organized in a different way such that look-ahead with FORCE would not be necessary to process the records in the desired order.

Organization of the Files: The two files for the billing application (Figure 7-16) can each contain two record types. The primary file (FIRST) contains a name and address record for every customer (record type 01) followed by the customer's charge record (record type 02) if any purchases were made during the month. The secondary file (SECOND) contains one record for every customer showing his previous balance (record type 03). If the customer is entitled to a discount, a record containing his discount rate follows the balance card. As you can see, both files are in ascending sequence according to a common match field containing the customer number.

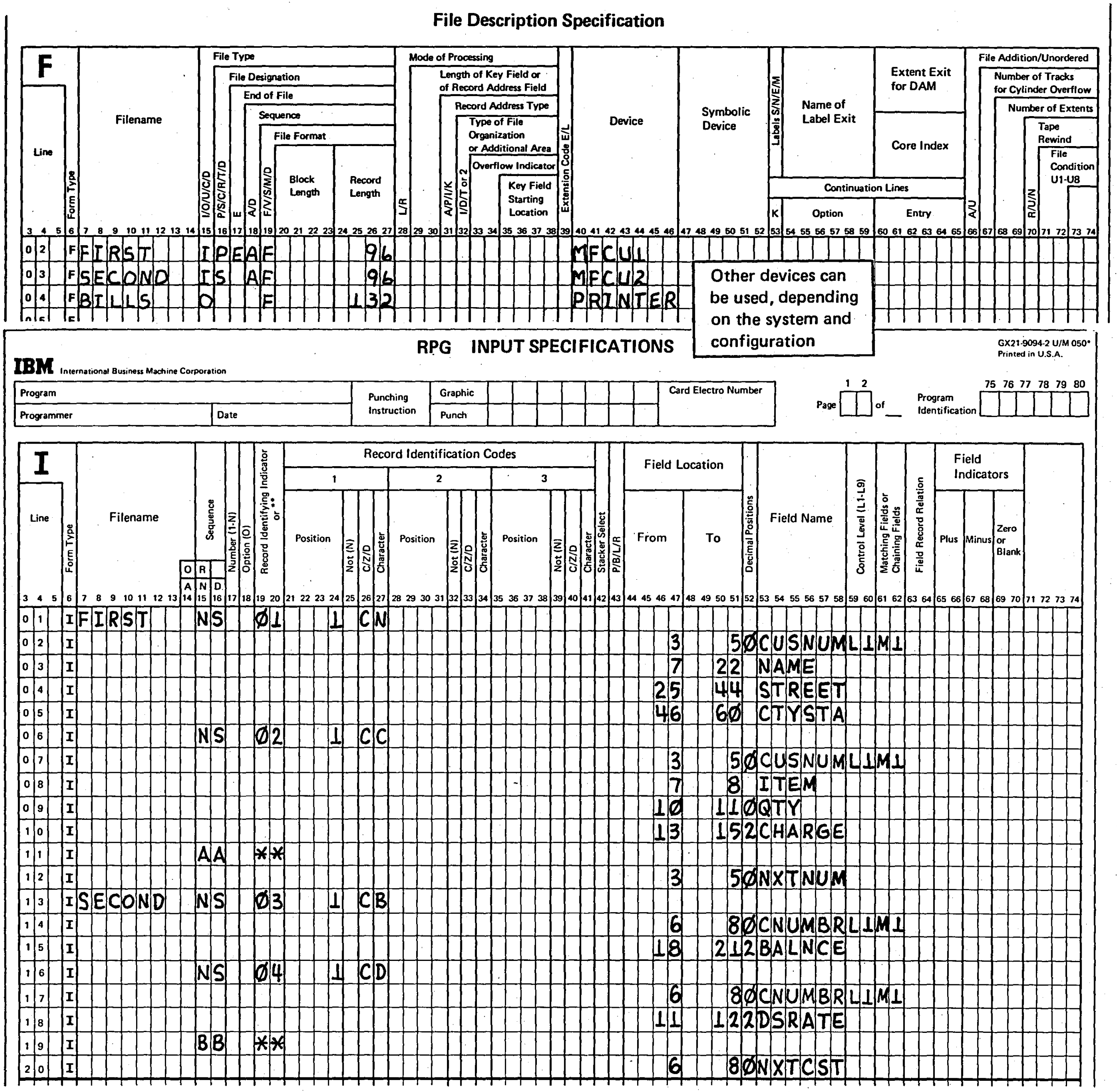

C 735 4C 01 998
N 735 GEST BRUCE 4856 NINTH STREET MPLS MN
C 631 5E 02 396
C 631 2B 04 276
C 631 3D 01 450
N 631 DEAN THOMAS 986 OAK GLEN DRIVE ST PAUL MN
N 563 CARLSON V 46 FIRST AVENUE HOPKINS MN
C 402 4B 03 297
C 402 3A 02 940
N 402 AKRE J 311 HAWK INS DRIVE BRAINERD MN
Record Type 02 (Charge Record)
Record Type 01 (Name and Address)
D 735 15
B 735 0000
B 631 6875
D 563 08
B 563 754
D 402 10
B 402 2986
Record Type 04 (Discount Record)
Record Type 03 (Previous Balance)

FIRST file

SECOND file

Figure 7-16. Files With More than One Matching Record

The specifications in Figure 7-17 define the files for this application. As entries in column 18 of the Input sheet show, record type 02 (charge records) in the primary file and record type 04 (discount record) in the secondary file may or may not be present for a particular customer. The use of the look-ahead fields defined on lines 11-12 and 19-20 will become clear as we explain the program.



File Description Specification

```
02 FFIRST   IPEAF      96            MFCU1
03 FSECOND  IS AF      96            MFCU2
04 FBILLS   O   F     132            PRINTER
05 F
```

Other devices can be used, depending on the system and configuration

IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094-2 U/M 050*
Printed in U.S.A.

```
01 IFIRST   NS  Ø1   1 CN
02 I                                     3   5ØCUSNUML1M1
03 I                                     7  22 NAME
04 I                                    25  44 STREET
05 I                                    46  6Ø CTYSTA
06 I        NS  Ø2   1 CC
07 I                                     3   5ØCUSNUML1M1
08 I                                     7   8 ITEM
09 I                                    1Ø  11ØQTY
10 I                                    13  152CHARGE
11 I        AA  **
12 I                                     3   5ØNXTNUM
13 ISECOND  NS  Ø3   1 CB
14 I                                     6   8ØCNUMBRL1M1
15 I                                    18  212BALNCE
16 I        NS  Ø4   1 CD
17 I                                     6   8ØCNUMBRL1M1
18 I                                    11  122DSRATE
19 I        BB  **
20 I                                     6   8ØNXTCST
```

● **Figure 7-17. Force With Look Ahead: Process Primary, Them Matching Secondaries Before Matching Primaries**

Determining the Order of Processing: To produce a bill, a customer's name and address are to be printed, followed by his previous balance (Figure 7-18). Any additional charges for the month are to be added to the previous balance to obtain a new balance. Some customers are entitled to a discount, which must be determined before adding the monthly charges to the previous balance.

If the two files are processed by RPG II logic according to matching records, all matching primary file records are processed before any matching secondary file records. In other words, first the name and address record is selected, followed by any charge records. Only after processing all the primary records for a particular customer are his balance and discount records processed.

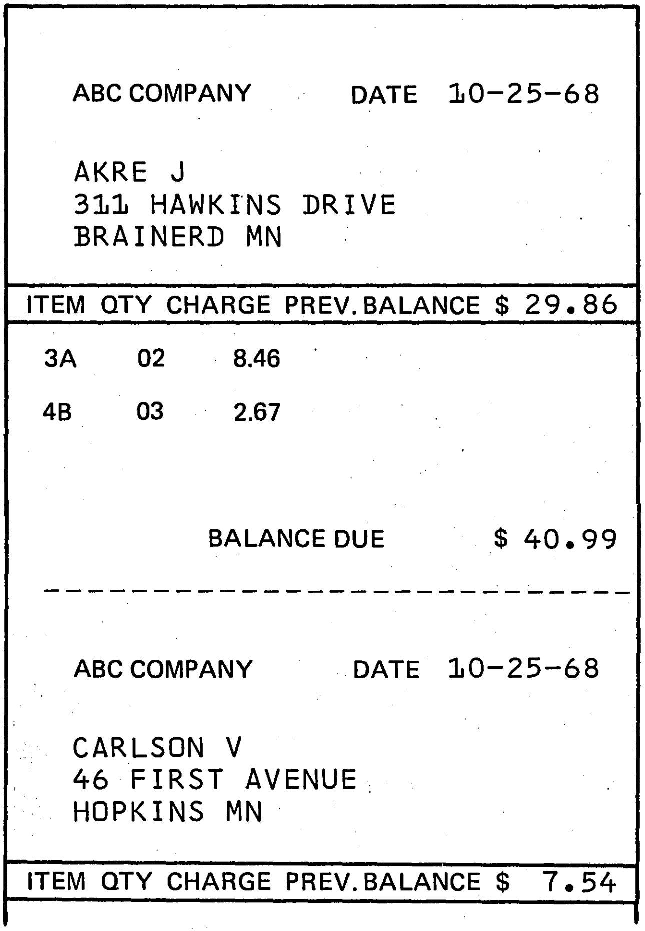
Providing a customer has no charge records, the order of processing determined by RPG II is correct. That is, after the name and address record (the only primary record with the same match field), the balance record from the secondary file is processed. If there is a discount record with the same match field, it is then read, even though the discount data is not necessary when there are no charges.

On the other hand, for customers who have incurred charges, the order of processing the matching records must be altered. Instead of processing all matching primaries before the matching secondaries, only the first primary record is processed. The matching secondary records are then processed, followed by the remaining matching primary records.

Determining When to Force A File: The calculation and output specifications to process the records and print the bills are shown in Figure 7-19. For every customer group, the name and address record (record type 01) from the primary file is read first and printed at the top of the billing form. RPG II logic automatically selects this primary record as the first to be processed.

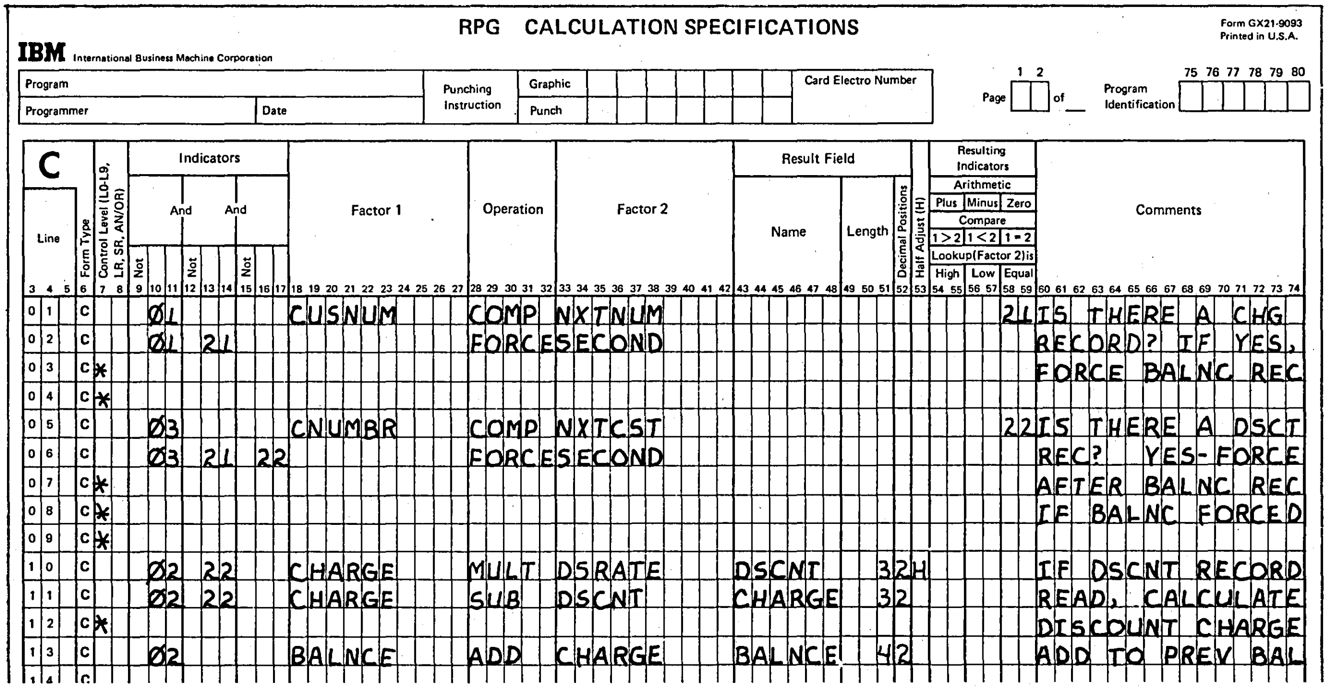
You want customer's balance record (record type 03) from the secondary file to be processed immediately after the name and address record. What you must determine then, while the name record (record type 01) is still being processed, is whether the balance record can be selected normally or if it must be forced. You know that if there is a charge record for this customer in the primary file, it has the same match field as the name and address record. Furthermore, if there is a charge record with the same match field, this record will be selected for processing rather than the secondary file balance record. Thus, while processing the name record, you must look-ahead at the next available primary record to determine if its match field is the same as the match field of the record being processed. To do this, line 01 of the calculation specifications (Figure 7-19) compares the look-ahead match field (NXTNUM) on the primary file record not yet available to the match field (CUSNUM) on the name record being processed. If the fields are equal (indicator 21 set on), you must force the balance record (line 02 of Calculation sheet).

While the balance record is being processed, you must determine from which file the next record is to be processed. To do this, calculation line 05 is performed. If there is a discount record in the secondary file for this customer, the match field on the next available secondary file record will be equal to the match field on the balance record being processed. Thus, the look-ahead match field (NXTCST) for the secondary file is compared to the balance record match field. If equal, indicator 22 is set on, indicating the next available secondary record (a discount record) should be processed. Whether to force the discount record depends on whether there are still primary file records which match. If there were charge records in the primary file, the balance record being processed was forced and, thus, indicator 21 is still on.



```
    ABC COMPANY          DATE  10-25-68

    AKRE J
    311 HAWKINS DRIVE
    BRAINERD MN

ITEM QTY CHARGE PREV.BALANCE $ 29.86
 3A   02   8.46
 4B   03   2.67

              BALANCE DUE       $ 40.99
-------------------------------------
    ABC COMPANY          DATE  10-25-68

    CARLSON V
    46 FIRST AVENUE
    HOPKINS MN

ITEM QTY CHARGE PREV.BALANCE $  7.54
```

Figure 7-18. Bill Showing Order in Which Data Must be Printed



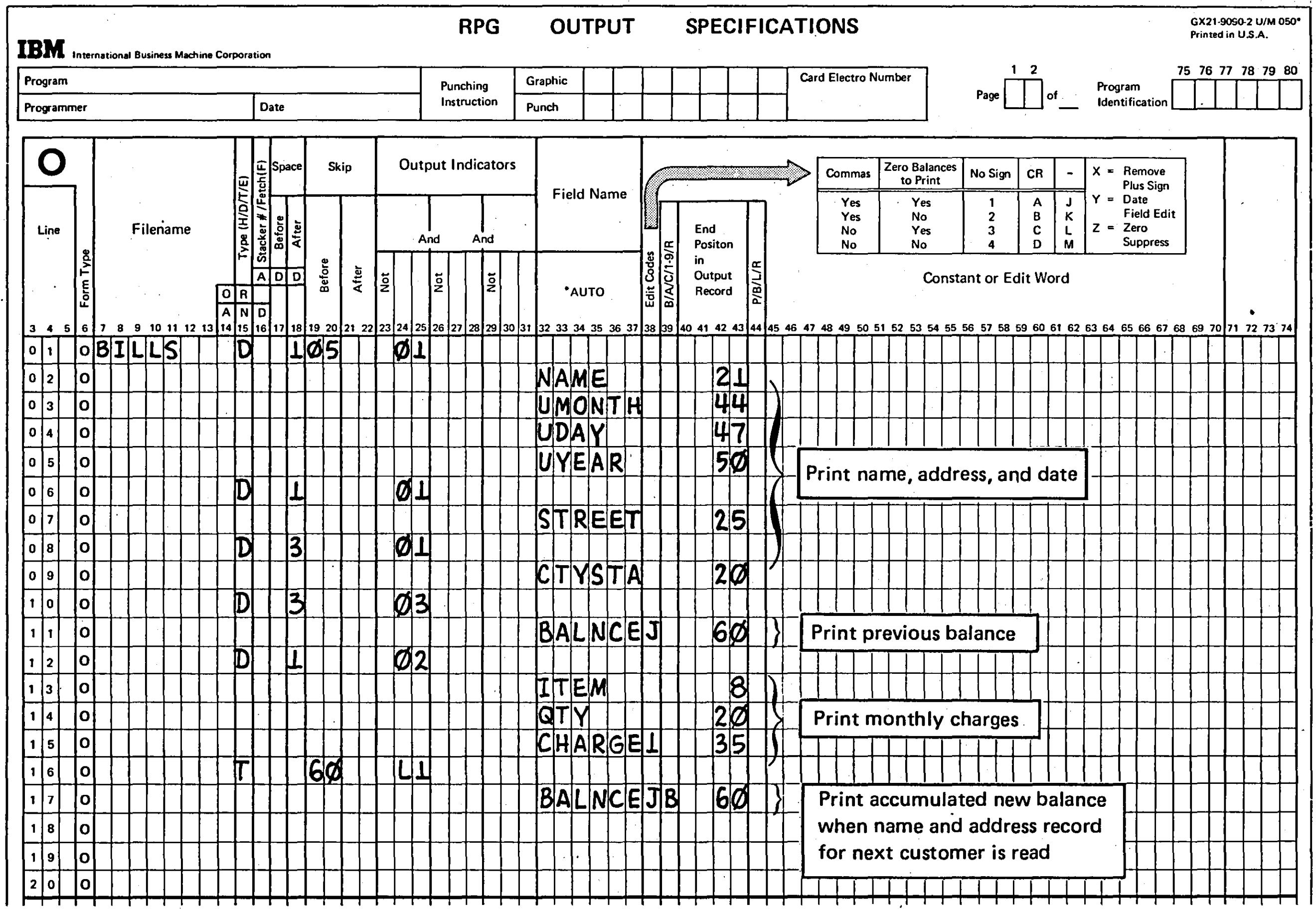
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	And	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High	Low	Equal	Comments
01	C		Ø1			CUSNUM	COMP	NXTNUM							21	IS THERE A CHG
02	C		Ø1	21			FORCE	SECOND								RECORD? IF YES,
03	C	*														FORCE BALNC REC
04	C	*														
05	C		Ø3			CNUMBR	COMP	NXTCST							22	IS THERE A DSCT
06	C		Ø3	21	22		FORCE	SECOND								REC? YES-FORCE
07	C	*														AFTER BALNC REC
08	C	*														IF BALNC FORCED
09	C	*														
10	C		Ø2	22		CHARGE	MULT	DSRATE	DSCNT	3	2	H				IF DSCNT RECORD
11	C		Ø2	22		CHARGE	SUB	DSCNT	CHARGE	3	2					READ, CALCULATE
12	C	*														DISCOUNT CHARGE
13	C		Ø2			BALNCE	ADD	CHARGE	BALNCE	4	2					ADD TO PREV BAL
14	C															



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090-2 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Skip Before	Skip After	Output Indicators	And	And	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	Constant or Edit Word
01	O	BILLS	D		1	Ø5		Ø1							
02	O										NAME			21	
03	O										UMONTH			44	
04	O										UDAY			47	
05	O										UYEAR			5Ø	
06	O		D		1			Ø1							
07	O										STREET			25	
08	O		D		3			Ø1							
09	O										CTYSTA			2Ø	
10	O		D		3			Ø3							
11	O										BALNCE	J		6Ø	
12	O		D		1			Ø2							
13	O										ITEM			8	
14	O										QTY			2Ø	
15	O										CHARGE	1		35	
16	O		T			6Ø		L1							
17	O										BALNCE	J	B	6Ø	
18	O														
19	O														
20	O														

Print name, address, and date

Print previous balance

Print monthly charges

Print accumulated new balance when name and address record for next customer is read

● **Figure 7-19. Specifications for Processing Matching Secondary Records Before Matching Primary Records**

The FORCE of a discount record on calculation line 06 is conditioned, then, by two resulting indicators: 22 on means there is a discount record to be processed and 21 on means that the discount record must be forced since the balance record was forced.

After determining which file is to be processed next and, then, whether that file is to be forced, output processing continues for the record being processed. That is, the balance is printed on the bill.

Providing a discount record is present, this record is processed next. No calculations and output are performed for this record, however. The discount rate (DSRATE) is merely stored so that data can be used to calculate discounts when charge records are processed.

In the next program cycle, RPG II logic determines which record to process next. If there is a charge record in the primary file, this record is selected since its match field is the same as the match field on the previously processed record.

When a charge record (record type 02) is selected, the first step is to determine if the charge is to be discounted. If a discount record was processed for this customer, indicator 22 is still on. Thus, the calculations on lines 10-11 are performed only if 22 is on. The charge, whether discounted or not, is then added to the previous balance to accumulate a new balance (line 13). The individual charge is then printed and any remaining charge records are processed in the same way.

When all charge records for this match group are processed, a name and address record for the next customer is available in the primary file, while his balance record is in the secondary file. As before, since both records match, RPG II logic selects the primary file name record to process first.

Referring to the input specifications for this program, the customer number field on the name record was assigned as a control field, as well as a match field (see Figure 7-17). Since the customer number on the name record just selected differs from that on the previous name record, a control break occurs. Before processing the name record for the second customer, then, total operations for the previous group are performed. Thus, the output specifications in Figure 7-19, line 13, cause the new balance (which has been accumulating for every charge record processed) to be printed on the customer's bill.

Effect of Forcing on the MR Indicator

In this last example, we assumed that for every customer, there is at least one record in the primary file (name record) and one record in the secondary file (balance record). For every record processed then, there exists a matching record in the other file.

Although a matching record condition actually exists for all records, the MR indicator is not necessarily turned on for every record. When RPG II logic selects records in the usual order, the MR indicator is on during the processing of these records. However, MR is never turned on for a record which is forced. If a file is to be forced, the RPG II logic doesn't even compare the two files and, thus, doesn't determine if a matching record condition exists. A forced record is processed, then, as if it contained no match fields.

The last example did not require that any calculations or output be conditioned on the basis of matching records. Nevertheless, you must be aware of the effect forcing has on the MR indicator so you won't incorrectly think MR is on when it is actually off.

PROCESSING DEMAND FILES (READ OPERATION)

Using the FORCE operation, you can override normal RPG II logic for selecting records on the *next* program cycle. Suppose, however, you want to select records to be processed during the *current* program cycle. For example, a company maintains a file of employee numbers, NUMBRFLE. Each number is on a separate record (Figure 7-20). Those records containing numbers that are already assigned to employees are identified by a flag (character X in position 8 of the record). If the number has not been assigned to an employee, the record does not have a flag.

For each record that is read from the file of new employee records, NEWNAME (Figure 7-20), one or more records must be read from NUMBRFLE to find a number that can be assigned to the new employee. The new number must be found during the current cycle, since the new employee record is added to the employee master file, NAMEFILE (Figure 7-20), after a number is assigned.

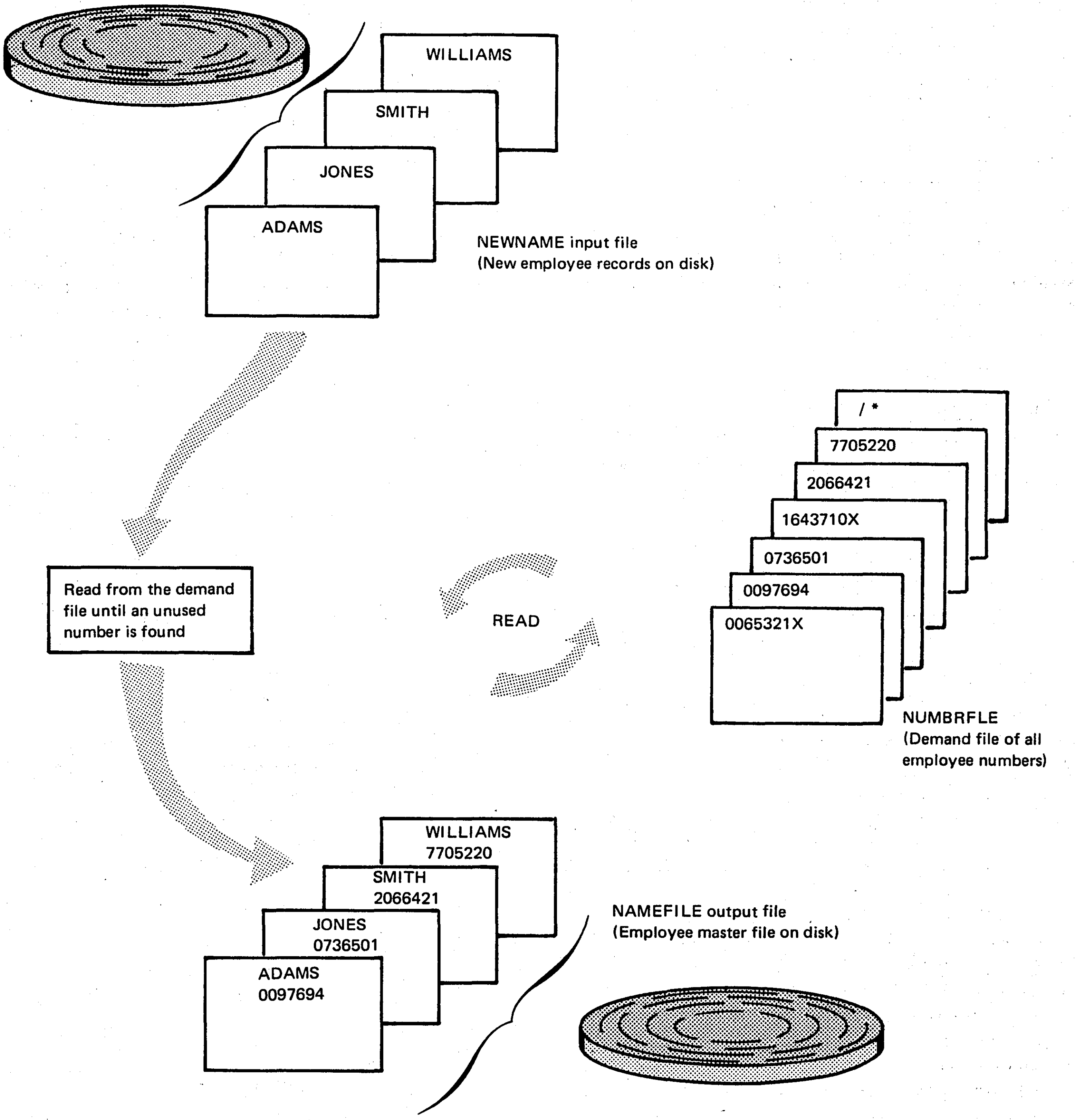
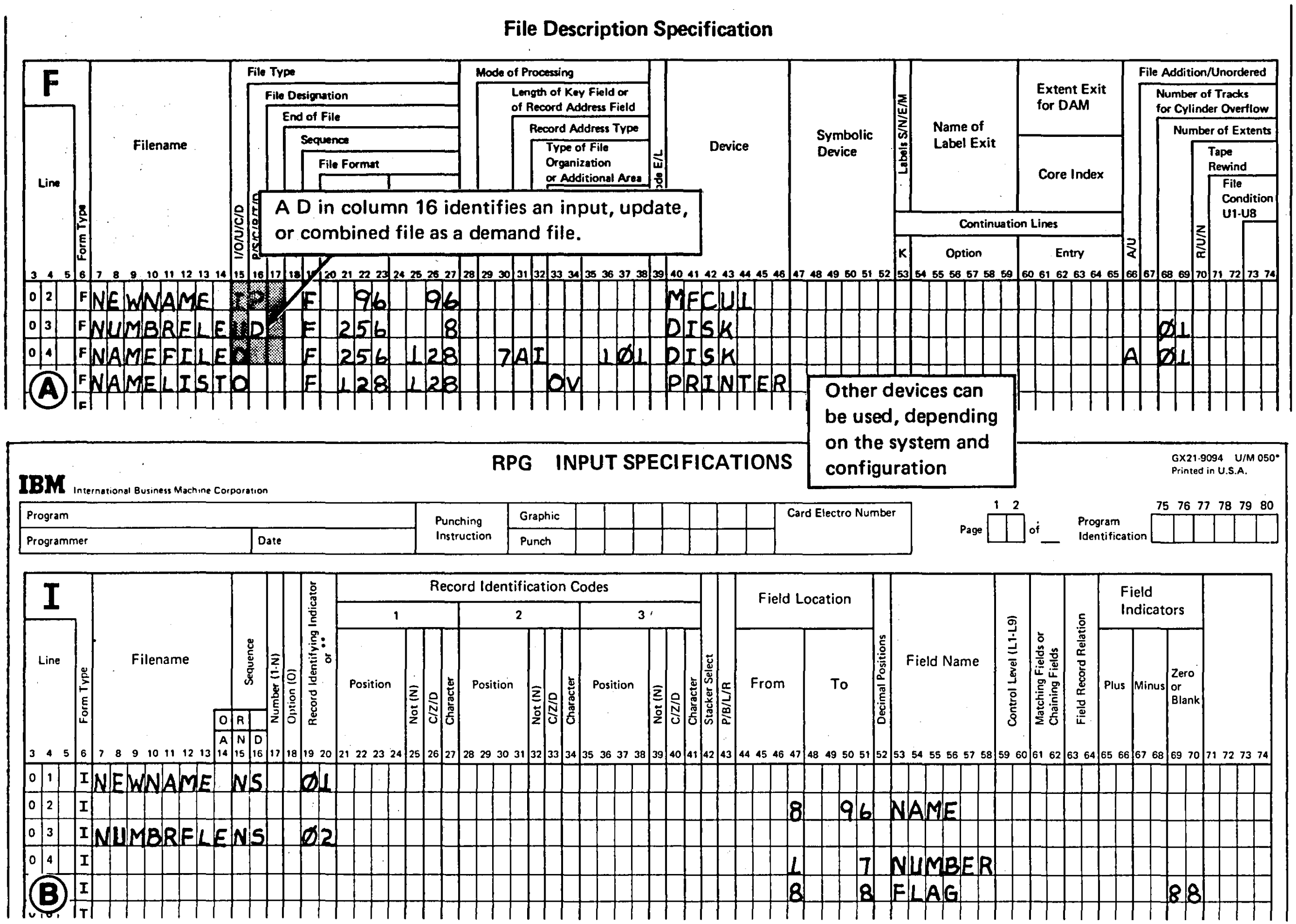


Figure 7-20. Using the READ Operation and a Demand File to Assign Man Numbers to New Employees

By designating NUMBRFLE as a *demand file* (Figure 7-21, insert A) you can request input from the file as many times as necessary during a single program cycle. In order to request input, you must use the READ operation code in the calculation portion of your program (Figure 7-21, insert C). Each time a READ operation is done, a record is read from NUMBRFLE.

It is possible that end of file for the demand file could be reached during a program cycle. Two options are available to handle this situation. In the example (Figure 7-21, insert C), an indicator has been entered in columns 58-59. An indicator specified in columns 58-59 of the specification line containing the READ operation will turn on after each READ operation if an end of file condition is reached. In the example, new employee records are listed on the printer as they are added to NAMEFILE. If end of file is reached on NUMBRFLE before numbers have been assigned to all new employees, indicator 77 is turned on and the new employee records to which numbers have not been assigned are listed with an appropriate identification.

If an end of file indicator is not entered in columns 58-59, the program will halt when end of file is reached on the demand file and each time READ is executed thereafter.



File Description Specification

A D in column 16 identifies an input, update, or combined file as a demand file.

```
02 FNEWNAME IP  F  96  96            MFCU1
03 FNUMBRFLEID  F 256   8            DISK                        Ø1
04 FNAMEFILEO   F 256 128 7AI    1Ø1 DISK                      A Ø1
   FNAMELISTO   F 128 128     OV     PRINTER
   F
```

Other devices can be used, depending on the system and configuration

IBM International Business Machine Corporation

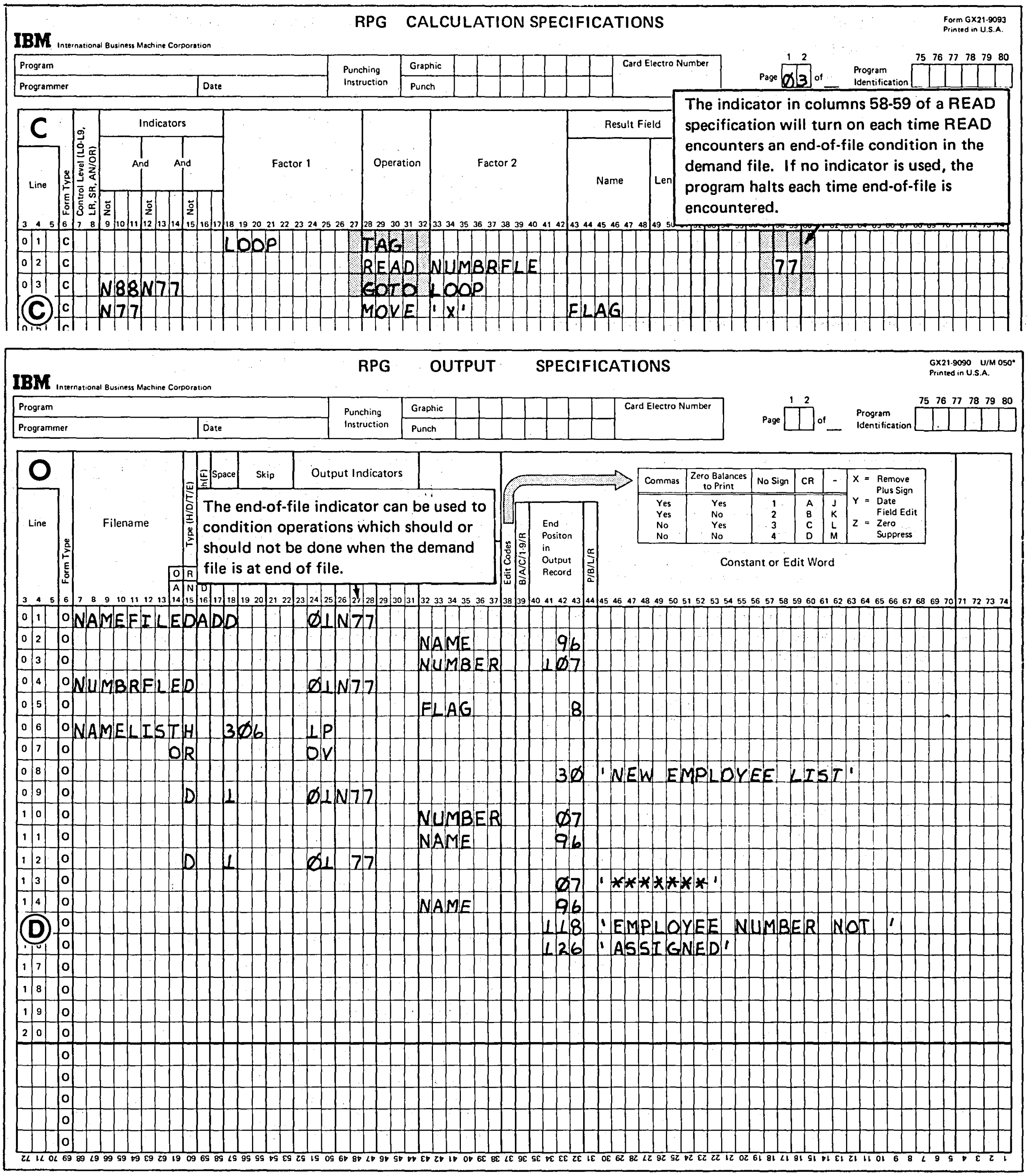
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page of
Program Identification

```
01 INEWNAME NS  Ø1
02 I                                        8  96 NAME
03 INUMBRFLENS  Ø2
04 I                                        1   7 NUMBER
   I                                        8   8 FLAG            88
   I
```

Figure 7-21 (Part 1 of 2). Coding to Use a Demand File to Assign Man Numbers to New Employees



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Page Ø3

```
01 C           LOOP      TAG
02 C                     READ NUMBRFLE                 77
03 C  N88N77             GOTO LOOP
04 C  N77                MOVE 'X'       FLAG
```

The indicator in columns 58-59 of a READ specification will turn on each time READ encounters an end-of-file condition in the demand file. If no indicator is used, the program halts each time end-of-file is encountered.

IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

```
01 ONAMEFILEDADD     Ø1N77
02 O                         NAME      96
03 O                         NUMBER   1Ø7
04 ONUMBRFLED        Ø1N77
05 O                         FLAG       8
06 ONAMELISTH 3Ø6    LP
07 O       OR        OV
08 O                                 3Ø 'NEW EMPLOYEE LIST'
09 O        D 1      Ø1N77
10 O                         NUMBER    Ø7
11 O                         NAME      96
12 O        D 1      Ø1 77
13 O                                 Ø7 '*******'
14 O                         NAME      96
15 O                                118 'EMPLOYEE NUMBER NOT '
16 O                                126 'ASSIGNED'
```

The end-of-file indicator can be used to condition operations which should or should not be done when the demand file is at end of file.

Figure 7-21 (Part 2 of 2). Coding to Use a Demand File to Assign Man Numbers to New Employees

Considerations for Using READ and Demand Files

The following files can appear as Factor 2 in a READ operation (all must be designated demand files with a D in column 16 of the File Description sheet):

1. Sequential disk files processed consecutively and specified as input or update files.

2. Indexed disk files processed sequentially by key and specified as input or update files.

3. Indexed disk files processed sequentially within limits and specified as input or update files.

4. Direct disk files processed consecutively as input or update files.

5. Console and CRT files specified as input files.

6. Card files specified as input or combined files.

7. Ledger card files specified as input or combined files.

8. Data recorder files specified as input files.

9. Tape input files.

10. SIOC input and update files (SPECIAL device).

11. Teleprocessing input and combined files (BSCA device).

12. Device independent input files.

When using the READ operation for demand files remember:

1. Demand files (except those assigned to the Model 6 KEYBORD) can only be processed by the READ operation.

2. Control levels, matching fields, and look-ahead fields are not allowed with demand files.

3. Numeric sequence testing on the Input sheet is not allowed for demand files.

4. The MR indicator may not be entered in columns 63-64 (Field Record Relation) on the Input sheet.

5. When a demand file is conditioned by a U1-U8 indicator which is not on, no records will be read from that file and the end-of-file indicator in columns 58-59 will not turn on.

6. When reading from several demand files during the same RPG II cycle, record identifying indicators assigned to the demand files will remain on throughout the cycle if the previous READ operations were executed successfully.

REPETITIVE OUTPUT (EXCPT OPERATION)

RPG II has a special operation code called EXCPT which allows you to write or punch as many records as are required during one program cycle.

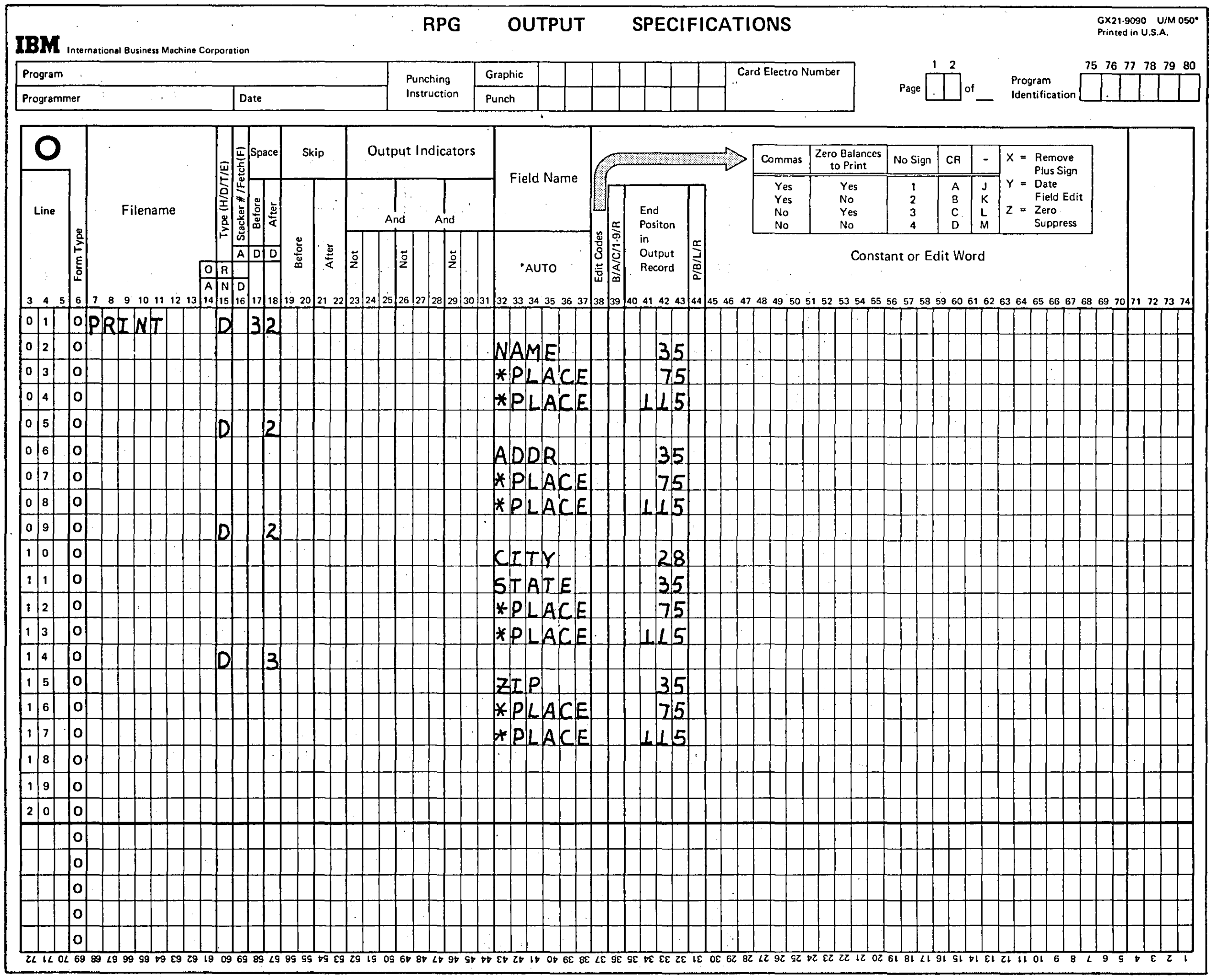
Normally a record is written or punched at either detail or total *output* time. Using EXCPT, records can be put out during detail or total *calculation* time. Each time you use the operation code EXCPT, specified records are written immediately. For example, if you use eight EXCPT operation codes in succession, you can get an exception output cycle eight times. The records are identical if the data fields in the exception records are not altered between the EXCPT operation codes on the Calculation sheet.

When you use the EXCPT operation code, you also must specify which records are to be put out during calculation time. These records are identified by an *E* in column 15 of the Output-Format sheet. Only those output lines identified by an *E* will be put out during an exception output cycle.

Using EXCPT and *PLACE

The reserved word *PLACE duplicates fields and places them on the same line. In the discussion of *PLACE in Chapter 3, an example is used in which three mailing labels were printed for each customer using *PLACE. If you wanted to print 15 labels for each customer, however, you could not use only the reserved word *PLACE. The only way would be to print the same three mailing labels five times in succession.

In the RPG II program cycle, each record specified is written or punched only *once* per cycle. For each record read by the program shown in Figure 7-22, the detail line specified in lines 01-04 is written only once. Remember that the *PLACE entry causes the field to be duplicated. Using *PLACE one line is printed with three identical names. The same is true for each of the other records specified. If you want to print 15 identical mailing labels, you need all records printed five times each.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

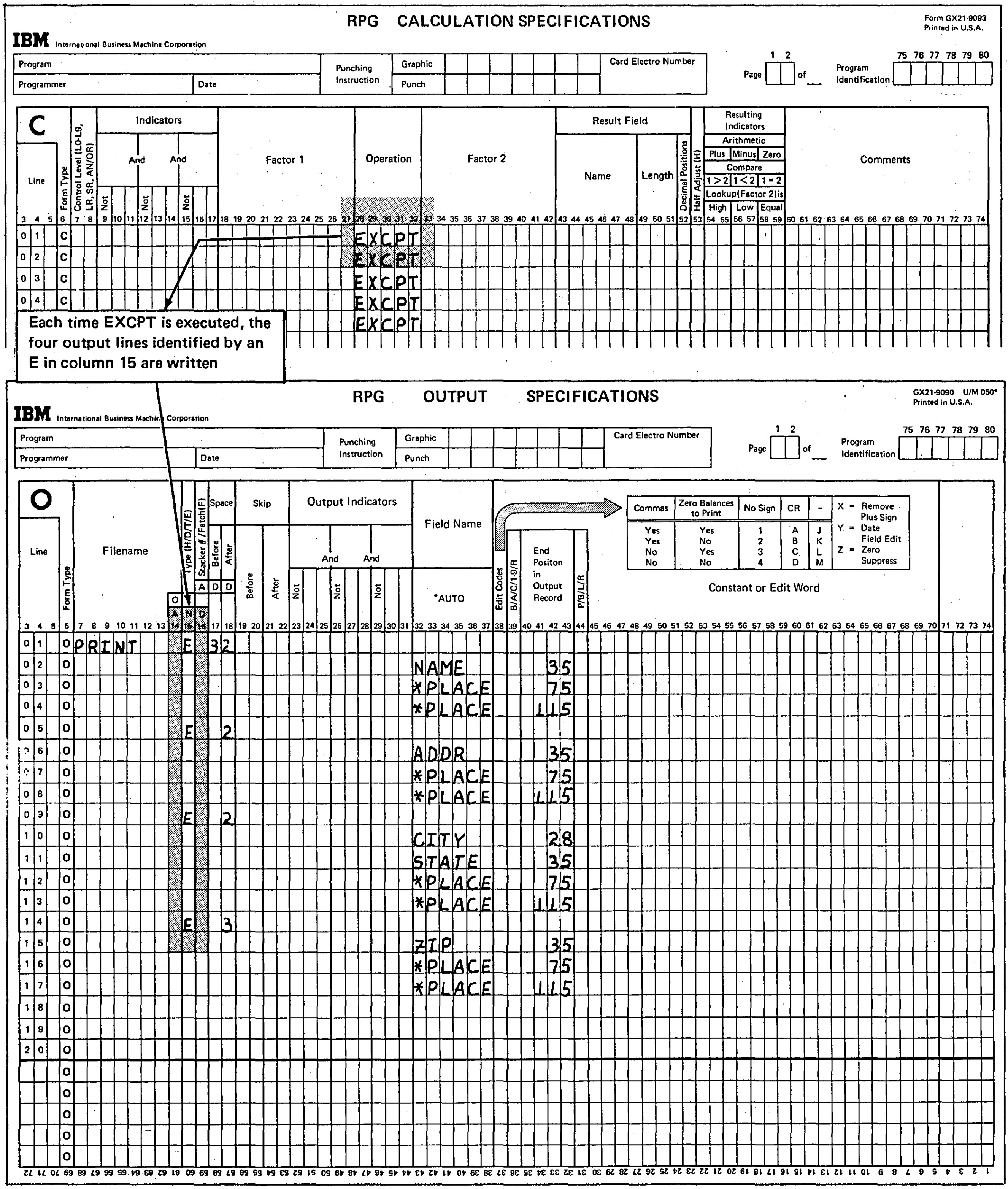
Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page of
Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Field Name	End Position in Output Record
01	O	PRINT	D	3	2		
02	O					NAME	35
03	O					*PLACE	75
04	O					*PLACE	115
05	O		D		2		
06	O					ADDR	35
07	O					*PLACE	75
08	O					*PLACE	115
09	O		D		2		
10	O					CITY	28
11	O					STATE	35
12	O					*PLACE	75
13	O					*PLACE	115
14	O		D		3		
15	O					ZIP	35
16	O					*PLACE	75
17	O					*PLACE	115
18	O						
19	O						
20	O						

Figure 7-22. Detail Output Operations



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Operation
01	C	EXCPT
02	C	EXCPT
03	C	EXCPT
04	C	EXCPT
		EXCPT

Each time EXCPT is executed, the four output lines identified by an E in column 15 are written

RPG OUTPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9090 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Field Name	End Position in Output Record
01	O	PRINT	E	3	2		
02	O					NAME	35
03	O					*PLACE	75
04	O					*PLACE	115
05	O		E		2		
06	O					ADDR	35
07	O					*PLACE	75
08	O					*PLACE	115
09	O		E		2		
10	O					CITY	28
11	O					STATE	35
12	O					*PLACE	75
13	O					*PLACE	115
14	O		E		3		
15	O					ZIP	35
16	O					*PLACE	75
17	O					*PLACE	115
18	O						
19	O						
20	O						

Figure 7-23. EXCPT Operation Code Used with Exception Records

Figure 7-23 shows the specifications necessary to print 15 mailing labels per customer. The *PLACE specifications on the Output-Format sheet will cause three mailing labels to be printed side by side on the paper. Each EXCPT code used on the Calculation sheet causes all records identified by an *E* in column 15 of the Output-Format sheet to be printed one time in the order shown on the sheet. Because all four lines are to be printed on the mailing label, all are identified by an *E*. The five EXCPT codes will cause five rows of three mailing labels each to be printed.

Another set will not be printed at detail output time, because all records having an *E* in column 15 can be printed only at calculation time when the EXCPT operation code is encountered.

EXCPT can be used with punched cards or disk as well as printed output. It operates in the same way in all cases. Each time the EXCPT code is encountered, output lines identified by an *E* in column 15 are executed.

Only output files may have EXCPT records specified; EXCPT cannot be used for combined files.

Conditioning the Use of EXCPT Operation

There are two ways you can condition an EXCPT operation: (1) on the Calculation sheet; and (2) on the Output-Format sheet.

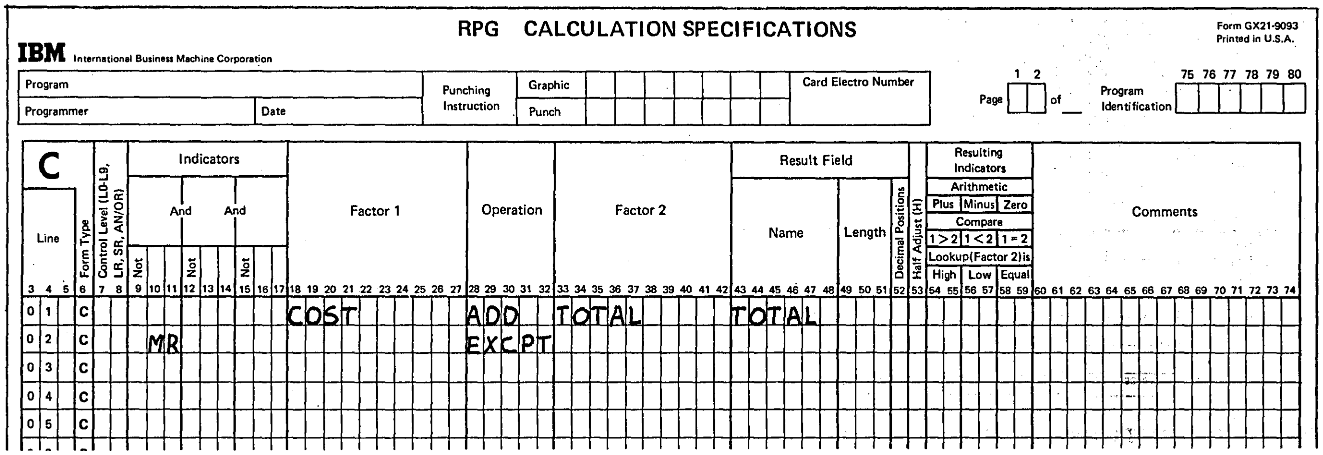
The EXCPT operation can be conditioned on the Calculation sheet in the same way as any other operation. As shown in Figure 7-24, the EXCPT records are put out only when MR is on.

An indicator used on the Calculation sheet controls the printing or punching of all EXCPT records. Individual EXCPT records are controlled by indicators specified in columns 23-31 of the Output-Format sheet. These indicators are used in the same way for EXCPT records as they are for all other records.

Restriction: Overflow indicators cannot be used to condition an EXCPT line. This means that an EXCPT record cannot be a record that is printed only when the end of the page has been reached.

Remember, these lines are exceptions. They print only at calculation time, not at output time. Therefore, they could /not possibly be printed when other overflow lines are.

An EXCPT line may be, however, printed on the overflow line. If it is, the overflow indicator will be turned on as usual. EXCPT lines can even *fetch* overflow. You may place an *F* in column 16 of any exception line. If the overflow indicator is on when the EXCPT line having an *F* in column 16 is reached, all lines conditioned by the overflow indicator will be printed before the exception line is printed.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program ______ Punching Instruction Graphic ______ Punch ______ Card Electro Number

Programmer ______ Date ______

Page 1 2 □□ of ___ Program Identification 75 76 77 78 79 80 □□□□□□

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus Minus Zero / Compare 1>2 1<2 1=2 / Lookup(Factor 2)is High Low Equal (54-59)	Comments (60-74)
01	C			COST	ADD	TOTAL	TOTAL					
02	C		MR		EXCPT							
03	C											
04	C											
05	C											

Figure 7-24. Conditioning the EXCPT Operation Code

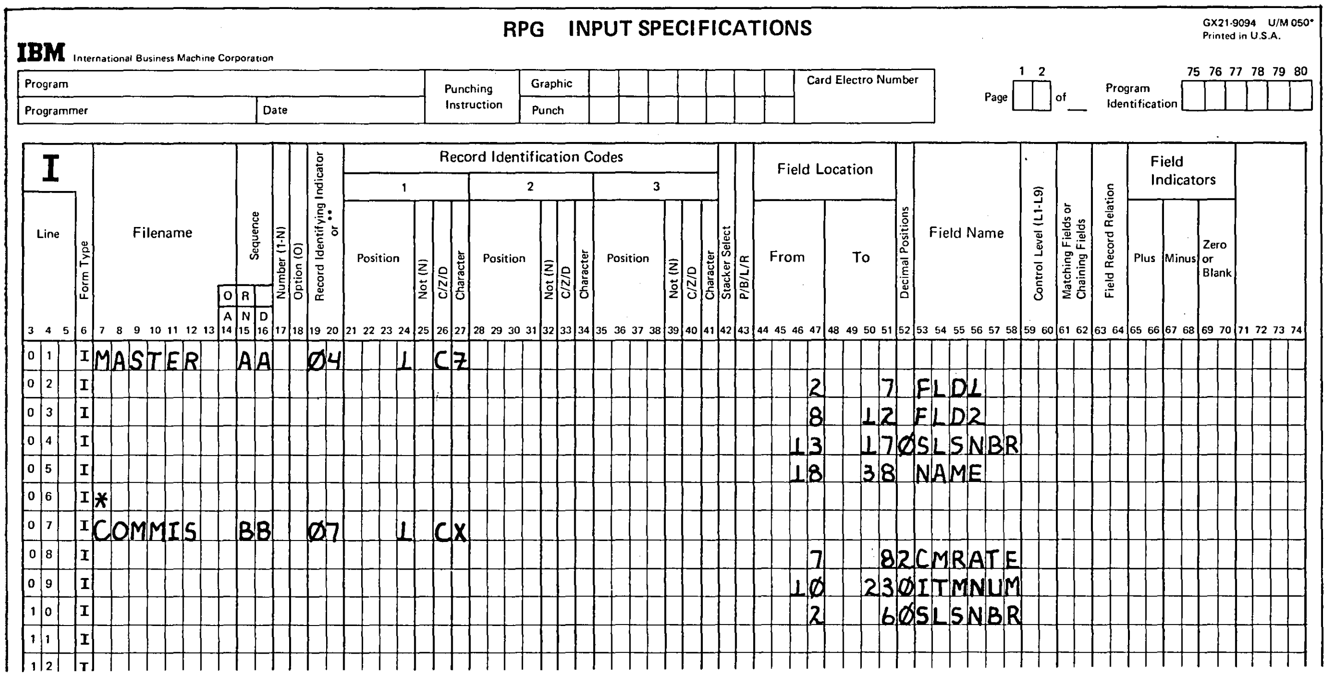
FORCE

1. What processing situations would require you to alter normal RPG II multifile logic by using the FORCE operation?

2. What two considerations are necessary to determine how the order of processing must be altered in a program?

3. Describe what occurs when a FORCE operation is performed.

4. A commission report is to be prepared listing each salesman and his commission amount. For each salesman record (MASTER file), there are seven commission records (COMMIS file). The records are described as follows:



RPG INPUT SPECIFICATIONS

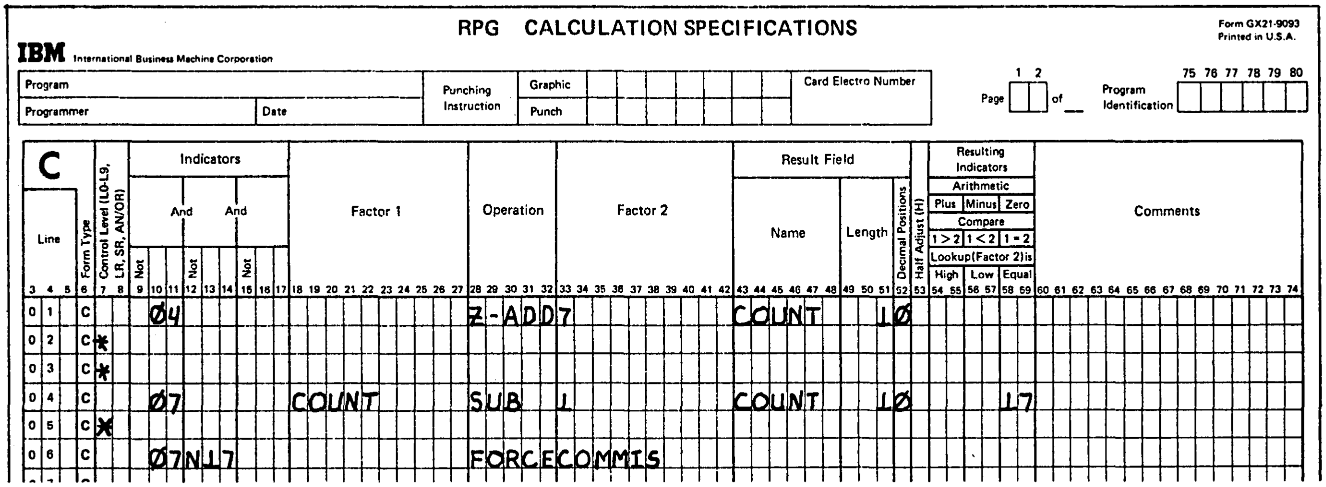
IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	C/Z/D	Character	From	To	Decimal Positions	Field Name
01	I	MASTER	AA	Ø4	1	C	Z				
02	I							2	7		FLD1
03	I							8	12		FLD2
04	I							13	17	Ø	SLSNBR
05	I							18	38		NAME
06	I*										
07	I	COMMIS	BB	Ø7	1	C	X				
08	I							7	8	2	CMRATE
09	I							10	23	Ø	ITMNUM
10	I							2	6	Ø	SLSNBR
11	I										
12	I										

a. Are the following calculation specifications correct for the required order of file processing?



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction | Graphic | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80
Programmer | Date | Punch

C

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators Plus / High / 1>2 (54-55)	Minus / Low / 1<2 (56-57)	Zero / Equal / 1=2 (58-59)	Comments (60-74)
01	C		Ø4		Z-ADD	7	COUNT	1	Ø					
02	C	*												
03	C	*												
04	C		Ø7	COUNT	SUB	1	COUNT	1	Ø				17	
05	C	*												
06	C		Ø7N17		FORCE	COMMIS								

b. On what condition will a record from MASTER be processed?

c. On what condition will a record from COMMIS be processed?

d. What specification changes must be made to process seven COMMIS records following each MASTER record?

e. Is FORCE necessary to process the records? Why?

5. Consider again the MASTER and COMMIS files as described in question 4, without match fields. However, assume that, for every MASTER record, there may be any number of related COMMIS records.

Using look-ahead fields and FORCE, code the necessary input and calculation specifications for determining the proper order of file processing. Compare the number of specifications to the number required if the files were processed by matching record logic.

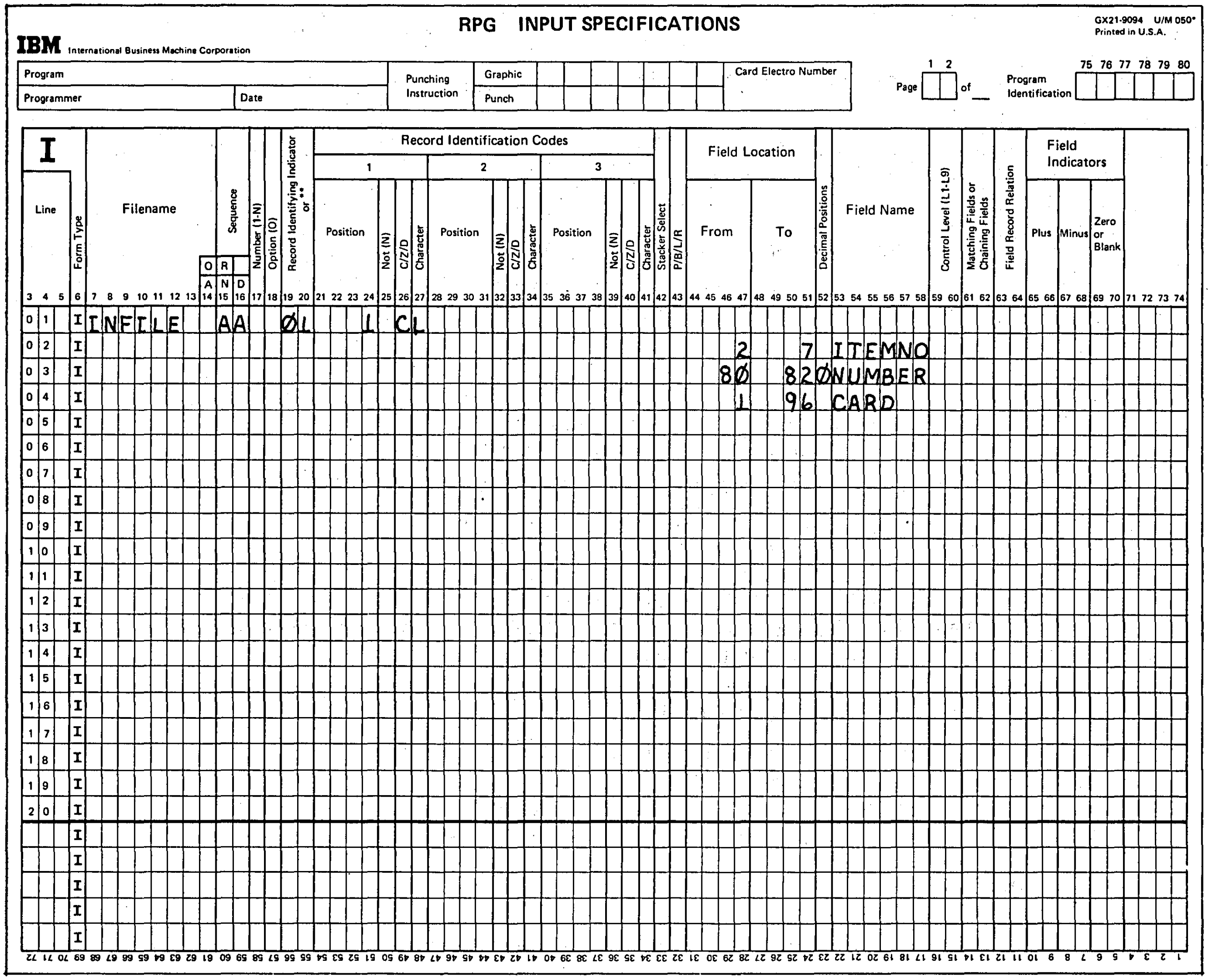
READ

6. List four ways in which a demand file differs from an ordinary secondary file.

EXCPT

7. What occurs when the EXCPT operation code is executed?

8. In a program, you need to punch a specified number of cards for each item. This number will be punched in each input card. Refer to the coded Input sheet for record layouts and code the Calculation and Output-Format sheets for the program.



RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

IBM International Business Machine Corporation

Program

Programmer | Date

Punching Instruction | Graphic | Punch

Card Electro Number

Page 1 2 □□ of __

Program Identification 75 76 77 78 79 80

I							Record Identification Codes														Field Location							Field Indicators			
							1				2				3																
Line	Form Type	Filename	Sequence OR AND	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Stacker Select	P/B/L/R	From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Plus	Minus	Zero or Blank	
3 4 5	6	7 8 9 10 11 12 13	14 15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I	INFILE	AA			ØL	1		C	L																					
0 2	I																				2	7		ITEMNO							
0 3	I																				8Ø	82	Ø	NUMBER							
0 4	I																				1	96		CARD							
0 5	I																														
0 6	I																														
0 7	I																														
0 8	I																														
0 9	I																														
1 0	I																														
1 1	I																														
1 2	I																														
1 3	I																														
1 4	I																														
1 5	I																														
1 6	I																														
1 7	I																														
1 8	I																														
1 9	I																														
2 0	I																														
	I																														
	I																														
	I																														
	I																														
	I																														

Answers To Review 7

1. a. Match fields cannot be assigned to the files and you need to:

   - Alternate processing between two files.
   - Process a primary file record followed by a number of secondary file records.
   - Process a secondary file record only when it matches a primary file record.

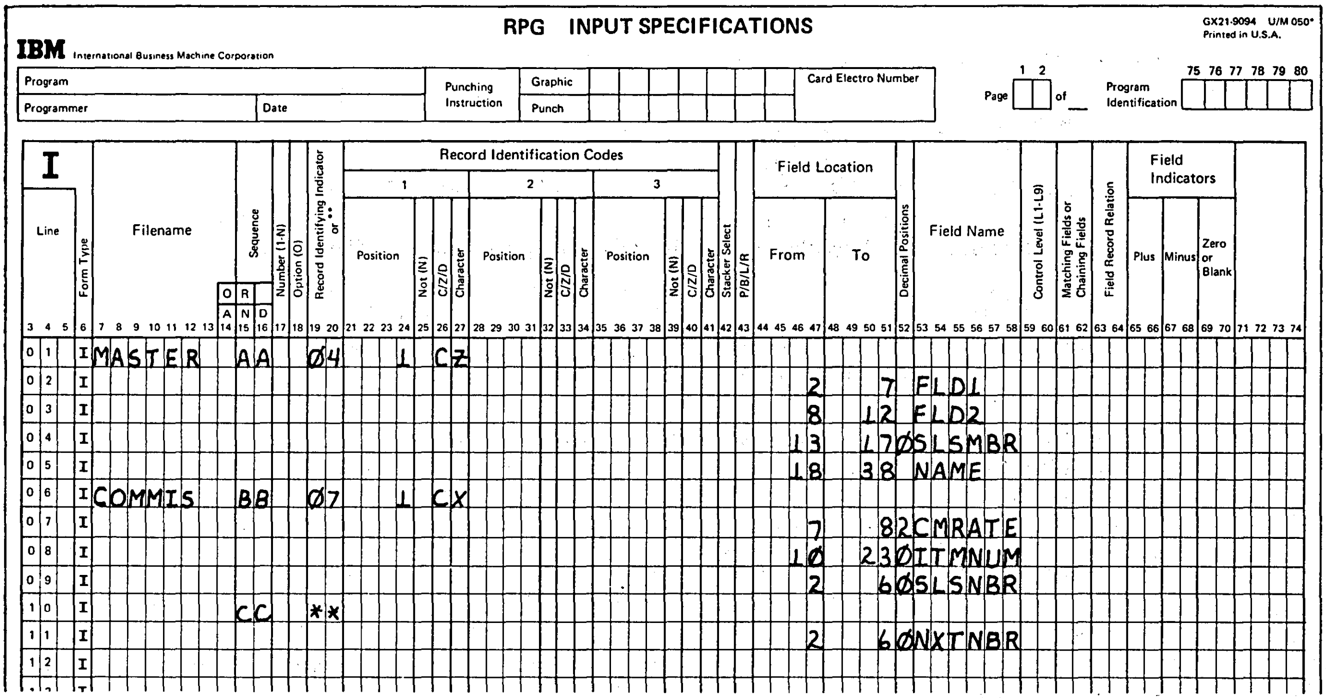
   b. Match fields are assigned to both files and you need to alter the order of matching record logic to process a primary file record, then matching secondary file records before matching primary file records.

2. a. When each file must be processed and under which conditions.
   b. Whether RPG II logic would select the appropriate record or if the file must be forced.

3. No action occurs at the time the specification is performed. At the beginning of the next program cycle, the next record from the file specified as Factor 2 of the FORCE operation is selected (by being forced) for processing.

4. a. No, the specifications are incorrect.
   b. A MASTER record will be processed in every program cycle until end of file is reached.
   c. A COMMIS record will not be processed until all records in the MASTER file have been processed.
   d. Lines 05 and 06 should not be conditioned by record identifying indicator 07. The COMP operation should be performed for every record type to determine if the FORCE is to be performed. It may be necessary to force a COMMIS record following either a MASTER record or another COMMIS record. For this reason, you must be able to perform the FORCE operation while processing either of the record types.
   e. The FORCE operation is not necessary. The RPG II logic of matching records can determine the proper order of processing if the SLSNBR fields on each record type are assigned as match fields on the Input sheet.

5. Input specifications to define look-ahead field:



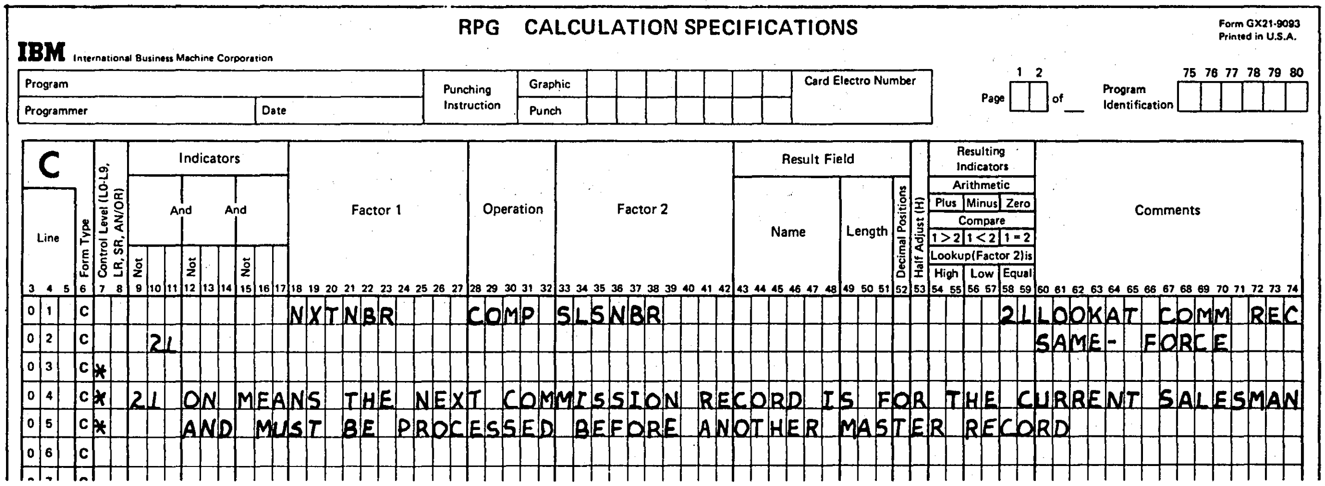
IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

```
01 IMASTER  AA  Ø4   1 CZ
02 I                                         2   7 FLD1
03 I                                         8  12 FLD2
04 I                                        13  17ØSLSMBR
05 I                                        18  38 NAME
06 ICOMMIS  BB  Ø7   1 CX
07 I                                         7   82CMRATE
08 I                                        1Ø  23ØITMNUM
09 I                                         2   6ØSLSNBR
10 I        CC  **
11 I                                         2   6ØNXTNBR
12 I
```

Calculation specifications to determine order of file processing:



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

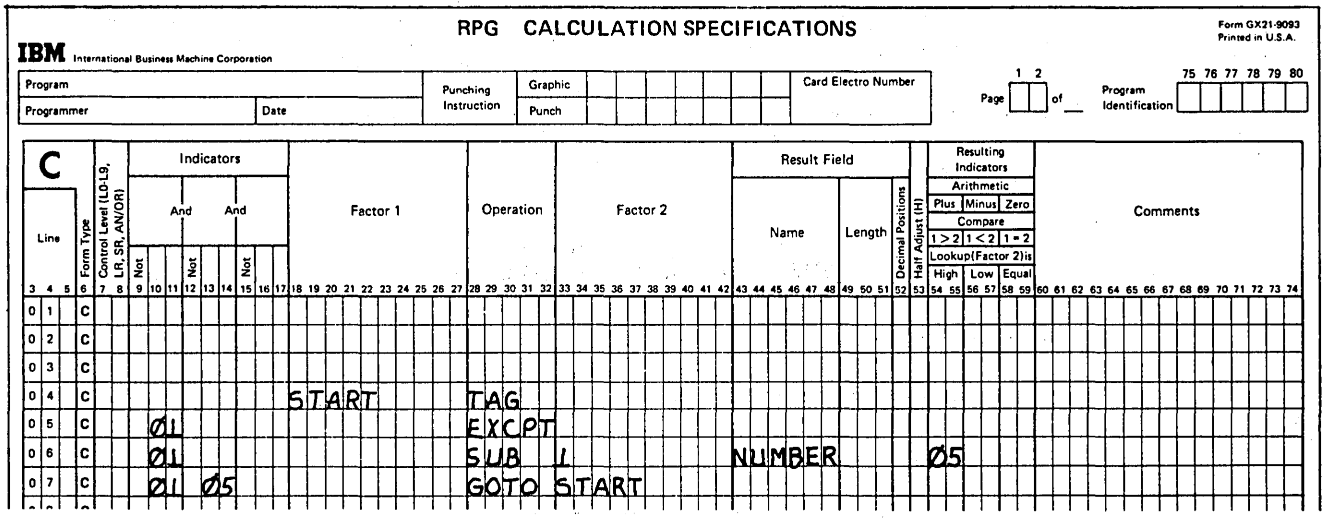
```
01 C           NXTNBR    COMP SLSNBR                   21LOOKAT COMM REC
02 C   21                                                SAME- FORCE
03 C*
04 C* 21 ON MEANS THE NEXT COMMISSION RECORD IS FOR THE CURRENT SALESMAN
05 C*    AND MUST BE PROCESSED BEFORE ANOTHER MASTER RECORD
06 C
```

If match field entries had been assigned to the SLSNBR fields on the Input sheet, input specification lines 10 and 11 would not be necessary to define the look-ahead field. No calculation specifications would be required to determine the order of processing.

6. a. A demand file is processed only by the READ operation code during calculations. Demand file records are never selected by normal RPG II multifile logic.
   b. Match fields cannot be assigned to a demand file.
   c. Look-ahead fields cannot be defined for a demand file.
   d. Reading an end of file record (/*) from a demand file does not set the LR indicator on. Instead, an end of file indicator can be entered in columns 58-59 of the READ specification line. This indicator will turn on each time a READ is executed which encounters an end of file condition in the demand file.

7. Immediate output for specified records occurs. These records are coded as exception records by an *E* in column 15 of the Output-Format sheet.

8. See Specification sheets.



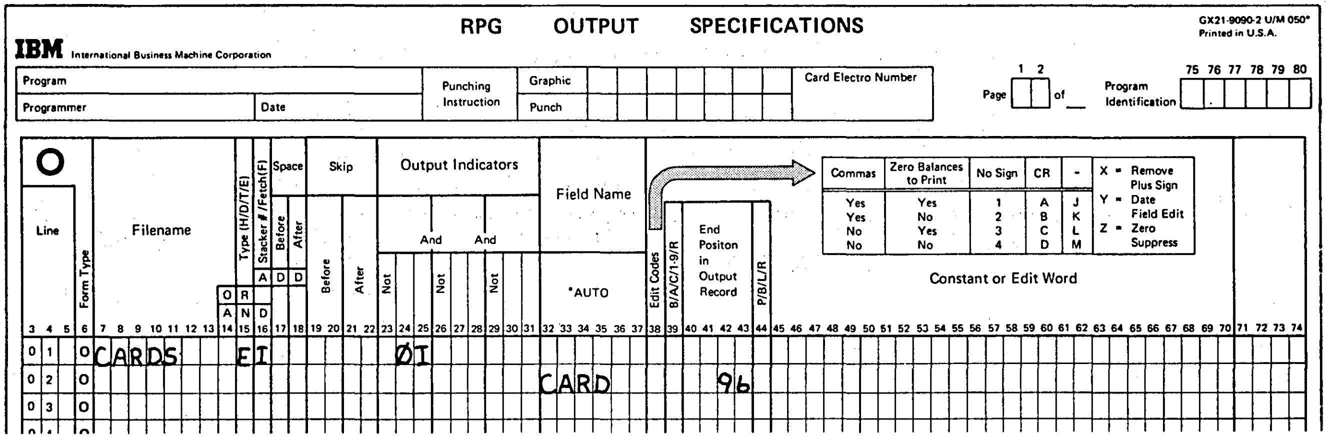
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic / Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Plus/High	Minus/Low	Zero/Equal	Comments
01	C													
02	C													
03	C													
04	C			START	TAG									
05	C		Ø1		EXCPT									
06	C		Ø1		SUB	1	NUMBER				Ø5			
07	C		Ø1 Ø5		GOTO	START								



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090-2 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic / Punch | Card Electro Number | Page of | Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker #/Fetch(F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name / *AUTO	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	CARDS	E	I					ØI						
02	O									CARD			96		
03	O														

Chapter 8. Tables

CHAPTER 8 DESCRIBES:

Uses for tables.

RPG II coding used to search tables.

Designing table input records.

LOKUP operation code.

Use of table data in calculations and output operations.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

RPG II object cycle.

Matching records processing.

Use of indicators to condition operations.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

State uses for tables.

Define a table on the Extension sheet.

Code problems using a single table.

Code problems using related tables.

Define and code the LOKUP operation code with tables.

Describe data and store it in a table.

Note: You can use the review questions contained in *Review 8* at the end of this chapter to test your comprehension of the chapter. Answers follow the review questions.

INTRODUCTION

If you wish to make a telephone call to a person, you must first determine his telephone number. Imagine trying to obtain the number if no telephone directories were available! For such reasons, similar items of information are grouped and organized so they can be referenced easily and quickly.

A *table* is a collection of related data organized in such a way that each item of information can be referenced by its position within the table. A telephone directory consists of two tables of information: a name list arranged alphabetically and a number list arranged in no apparent order. Each telephone number, however, occupies a position in the number list corresponding to the position of a particular name in the name list.

Each item within a table is called a table *element*. Thus, each name would be an element of the name table, while each number would be an element of the telephone number table.

If you wished to determine Ken Adams' telephone number, you would look through the list of names to locate *KEN ADAMS.* This procedure of checking the elements of a table one at a time to find a particular entry is called *searching* a table. Before looking through the name list, you must know what information you wish to find, the name *KEN ADAMS.* This data is referred to as the *search word.*

As shown in Figure 8-1, the name list is searched to find an entry which is equal to the search word. The matching entry, *KEN ADAMS,* is found in the third element of the name table. His corresponding telephone number, then, is found by selecting the third element in the number table.

When two related tables are used, as in a telephone directory, actually only one table is searched (name table). When the search condition (in this case, an equal match) has been satisfied, the data in the corresponding element of the second table becomes available. Thus, the first table is used as a means of locating data in the second table.

A telephone directory is an example of tables which organize information that we must reference over and over again in our every day lives. Likewise, tables can be used to organize data which must be referenced repeatedly in your data processing jobs.

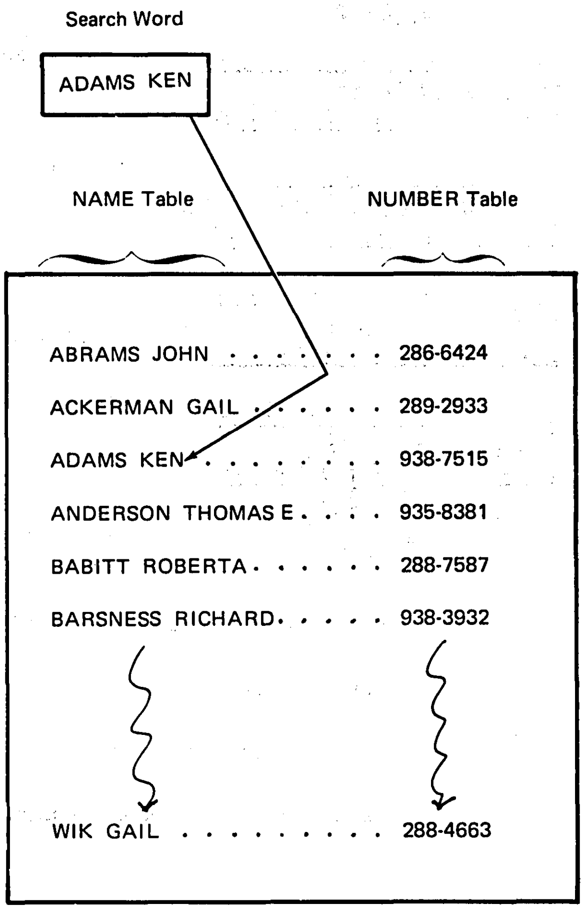

Search Word
ADAMS KEN
NAME Table
NUMBER Table
ABRAMS JOHN 286-6424
ACKERMAN GAIL 289-2933
ADAMS KEN 938-7515
ANDERSON THOMAS E 935-8381
BABITT ROBERTA 288-7587
BARSNESS RICHARD 938-3932
WIK GAIL 288-4663

Figure 8-1. Searching a Table

Let's assume that customers have purchased various items from a company sales catalog. The sales file (Figure 8-2) would contain records showing the customer's account number (CUSTMR), the item ordered (ITMORD), identified by a code, and how many were ordered by that customer (QTYORD).

Furthermore, the company keeps an inventory file (Figure 8-2) to contain data about each item which is carried in stock. A separate record is kept for each item showing the item code (ITEM), the quantity on hand (QTYSTK), and the unit cost of each item (COST).

Before you can ship the customer's orders, you must first determine if the item ordered is still carried in stock. To do this, a clerk could spend time looking up each item ordered to see if that item is recorded in the inventory file. However, the same item will probably be ordered by many customers. Thus, the same inventory file records would have to be referenced over and over again.

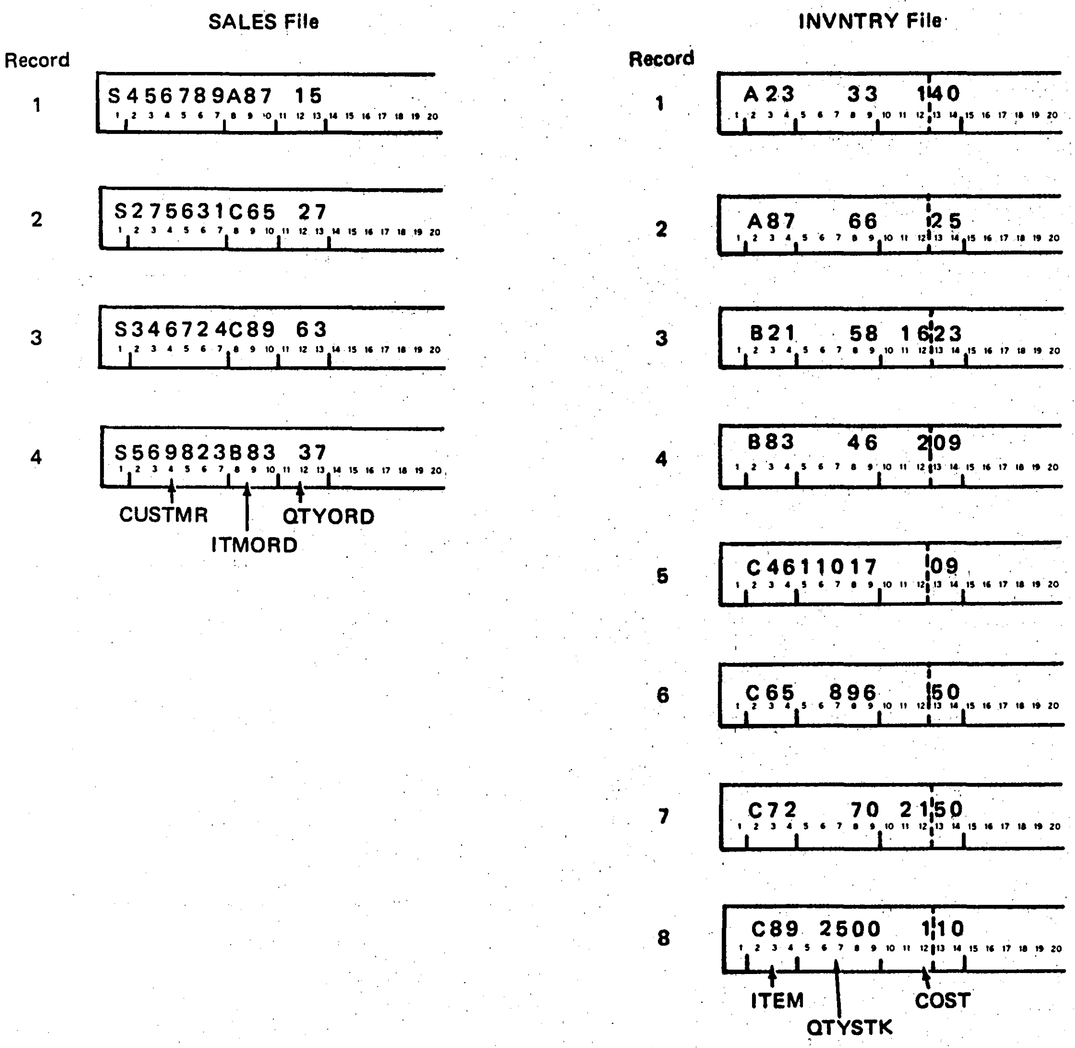


Figure 8-2. Data for Determining if Orders Can be Filled

Searching a Single Table

RPG II can search for the data in much less time by performing a table lookup function. As shown in Figure 8-3, a table (TABITM) would be set up in storage to contain all of the items available (see *Loading Tables* in this chapter for methods of loading table data into storage).

The second field of each sales record tells the program which item to *look up*. For every sales record read into the computer, TABITM is checked to see if the search word (item code on the sales record) matches an entry in the table.

In addition to searching for data quickly, use of the table lookup can often reduce the number of RPG II specifications needed in a program. All you must do is set up and define the table, and specify that the lookup operation is to be performed.

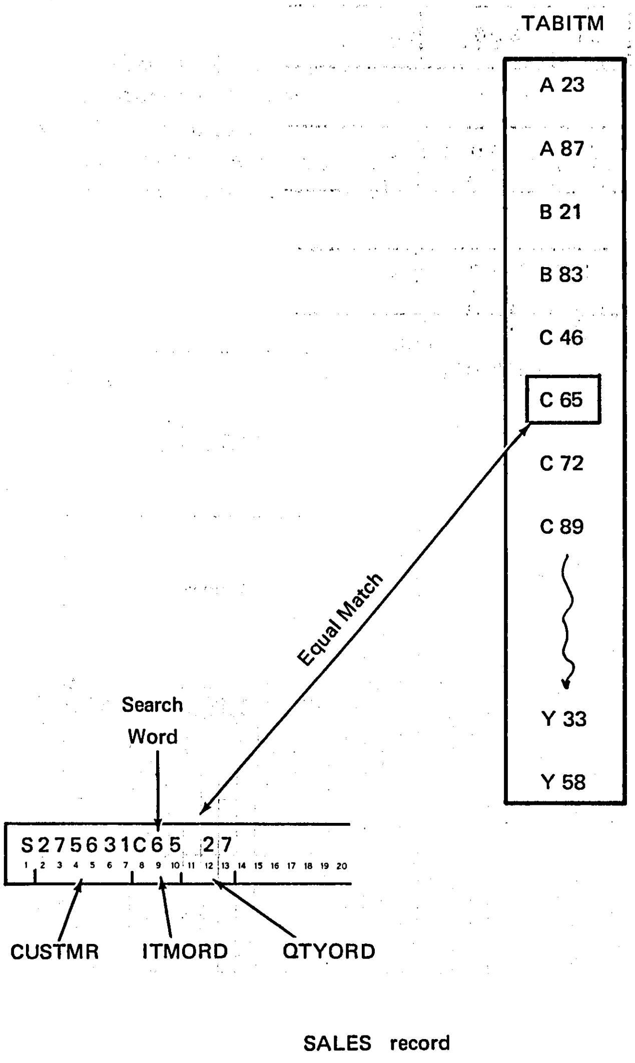


Figure 8-3. Searching a Table for a Particular Data Item

Designing Table Input Records

Data used to create a table must be obtained from table input records. These records can be taken from the keyboard or CRT, punched cards, magnetic tape, or disk. The records are read by the computer and entries are placed side by side in storage to form the table.

The inventory file for this company indicates that 98 types of items are carried in stock. Therefore, the 98 item codes must be contained in table input records to provide the entries for TABITM.

Number of Table Input Records Required for a Table

How many records would be necessary to punch the 98 item codes? That is entirely up to you. The number of records required to contain the table data depends on the number of entries you want on each record.

Number of Entries on a Table Input Record

Table input records may either contain one entry or a number of entries. The point to remember is that all table input records for a single table must contain the same number of entries, except for the last record.

The use of one entry per record is very convenient if it is necessary that the entries be in a particular order. Then to add or delete entries, you merely add or remove a record. Otherwise, you would have to recreate all of the records from the point of change to the end of the table. However, in the TABITM table, item code entries need not be in order, so including a number of entries in each record reduces the number of records required.

For TABITM, 98 entries must be recorded on table input records. If 96-column cards are used, 32 entries fit on each record, since each item code is three characters long. Therefore, to save card space, you could punch three table input records containing 32 entries each and a fourth table input record for the remaining two entries. (See Figure 8-4.) In this way, all records contain the same number of entries except the last record.

Of course, you do not have to fill the entire record with entries, as we have done. For instance, you may want the last 36 columns of a card to contain no punches. In that case, you could punch 20 entries (card columns 1-60) into each of four table input records. The remaining 18 entries could then be punched into a fifth record.

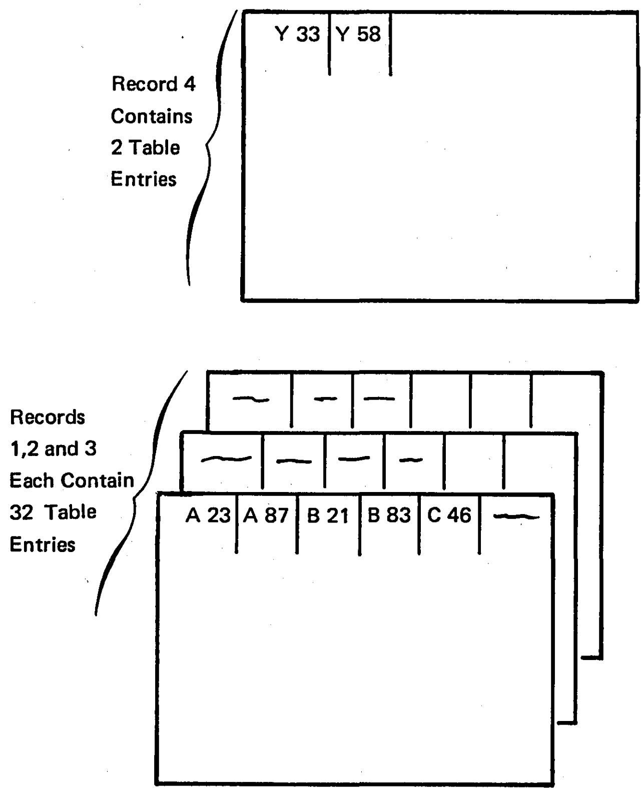


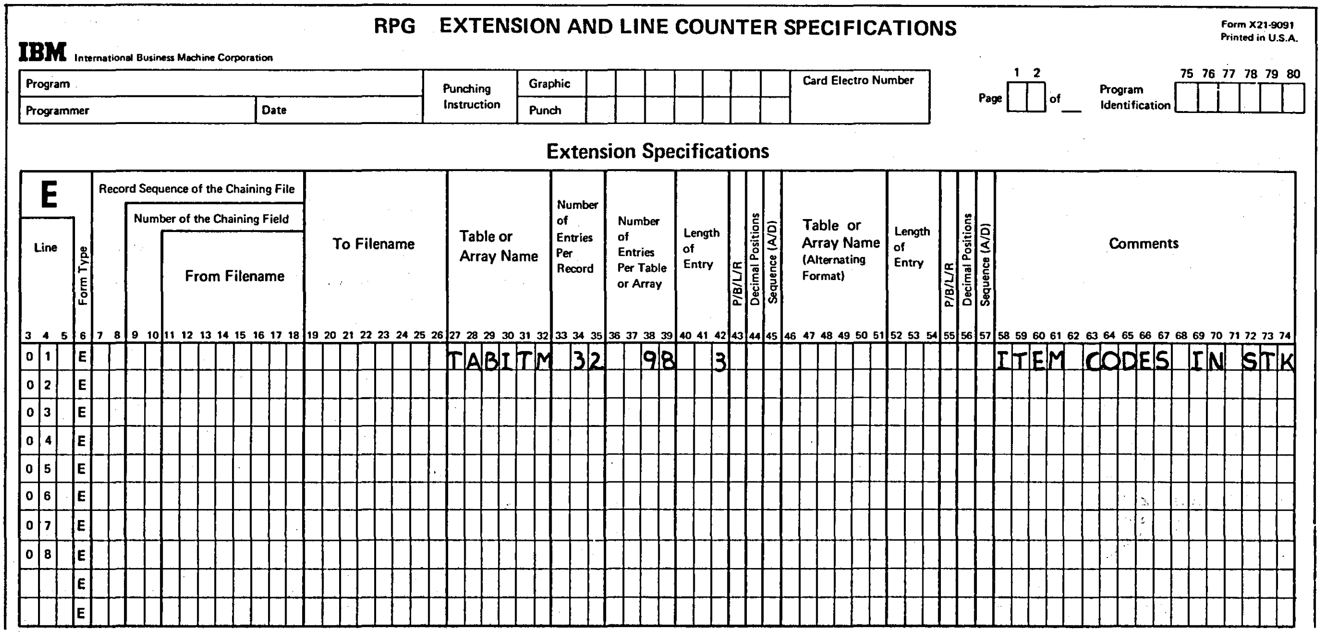
Figure 8-4. Four Table-Input Records for TABITM Entries

Notice in Figure 8-4 that the two entries in the fourth table input record are placed side by side. Since there are unused card columns on the record, why not space the entries? You cannot, because all entries must be continuous on the record, with no blank columns *between* entries. Furthermore, the first entry on each record must begin in position one.

Describing Table Input Records with Extension Specifications

Once the table input records have been designed, you then describe them to the RPG II Compiler program. Ordinarily, the data on input records is described by entering specifications on the Input sheet. However, you describe the data on table input records by coding extension specifications on the upper half of the Extension and Line Counter sheet. (Hereafter, we will refer to the upper portion of the form as the Extension sheet and the lower portion as the Line Counter sheet.)

Figure 8-5 shows the extension specifications needed to describe the table containing item codes. As you can see, a single table can be described using only columns 27-45. Of course, remember that the page and line number (columns 1-5) and form type (E in column 6) are also part of the entire specification line.



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program
Programmer
Date

Punching Instruction: Graphic, Punch

Card Electro Number

1 2
Page □□ of __

75 76 77 78 79 80
Program Identification □□□□□□

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E					TABITM	32	98	3									ITEM CODES IN STK
0 2	E																	
0 3	E																	
0 4	E																	
0 5	E																	
0 6	E																	
0 7	E																	
0 8	E																	
	E																	
	E																	

Figure 8-5. Describing Table-Input Records

The extension specifications provide the following information about each table to be used:

1. Name of each table (columns 27-32).

2. Number of table entries per table input record (columns 33-35).

3. Number of table entries per table (columns 36-39).

4. Length of each entry (columns 40-42).

5. Whether packed or binary data is contained in the table (column 43).

6. Number of decimal positions in each numeric entry (column 44).

7. Sequence of table entries, if any (column 45).

Each type of entry will be discussed in turn. From Filename (columns 11-18), To Filename (columns 19-26), and the entries in columns 46-57 are described later in this chapter.

Assigning Table Names

Every table used in a program must be assigned a name from three to six characters long. The table name may contain any combination of alphabetic characters and numbers. However, the first three characters of the name must be TAB.

With these points in mind, which of the following table names are *not* acceptable and why?

TABA	15ABCD
TAB C	TABSTATE
TB123	*TAB
TAB$2	

TAB C and *TAB are not acceptable, as table names cannot contain blanks or special characters, such as the *. TAB$2, on the other hand, is an acceptable table name, since $ is one of the three special characters which can be considered an alphabetic character. (The other two are # and @.)

The names 15ABCD and TB123 are invalid because they do not begin with the alphabetic characters TAB. Since TABSTATE contains more than six characters, it cannot be an acceptable table name. TABA and TAB$2 are the only two valid table names shown.

If possible, it is helpful to assign table names which are meaningful. For this example, the name TABITM has been assigned. Such a name gives an indication of what kind of data is contained in the table (in this case, item codes).

When a single table is being used, as in this case, the table name is entered in positions 27-32 of the Extension sheet (see Figure 8-5).

Number of Table Entries per Table Input Record

The number of entries in each table input record is specified in columns 33-35. Figure 8-5 shows 32 entries per record for TABITM. In this way, the compiler program will expect all table input records to contain 32 entries, except the last record which may have fewer entries. Notice that the number entered in these columns should end in column 35.

Number of Table Entries Per Table

The number of entries which can be contained in the entire table is entered in columns 36-39 of the Extension sheet. As shown in Figure 8-5, the number (98 for TABITM) should end in column 39.

Length of an Entry

The length of each table entry is indicated in columns 40-42, with the number ending in column 42. Numeric table entries may be up to 15 digits long, while alphameric entries can be as long as the maximum record length for the device (256 maximum).

For TABITM, the length *3* has been entered. It is possible to specify only one length. Therefore, this necessarily means that all entries in a table must be the same length.

At this point, you may be wondering what to do if all entries are not the same length. For instance, you might wish to make a table containing the months of the year. The solution is simple – all entries are made to be the length of the longest entry. The word SEPTEMBER contains the most characters. Therefore, each entry should be 9 characters long. To make JUNE an entry with length of 9, you would place 5 blanks after the letter E (see Figure 8-6). Inserting extra blanks to lengthen the data entry is referred to as padding with blanks. If your table entries were numbers, instead of letters, you would pad the short entries with zeros or blanks. (For numeric entries, the zeros or blanks would probably be placed in front of the number.)

```
J A N U A R Y ƀ ƀ
F E B R U A R Y ƀ
M A R C H ƀ ƀ ƀ ƀ
A P R I L ƀ ƀ ƀ ƀ
M A Y ƀ ƀ ƀ ƀ ƀ ƀ
J U N E ƀ ƀ ƀ ƀ ƀ
J U L Y ƀ ƀ ƀ ƀ ƀ
A U G U S T ƀ ƀ ƀ
S E P T E M B E R   ←—— Longest Table Entry
O C T O B E R ƀ ƀ
N O V E M B E R ƀ
D E C E M B E R ƀ
```

Table Of Months

Figure 8-6. Making Table Entries the Same Length

Padding a table entry with blanks should not be confused with spacing entries on a record. As mentioned, no blanks can occur *between* entries. However, blanks can be part of an entry in order to make all entries the same length.

Packed or Binary Table Data (Systems with Disk Storage)

An entry must be made in column 43 if the data for a pre-execution time table is in packed decimal (P) or binary (B) format on disk or tape or in packed format on 80-column cards (not allowed on 96-column cards). This entry applies only to numeric tables. For numeric tables in packed decimal format, the unpacked decimal length of the entries must be entered in columns 40-42 (Length of Entry). For numeric tables in binary format, enter the number of bytes required in storage for the binary field. For a two-position binary field, the entry in columns 40-42 is 4; for a four-position binary field, the entry is 9.

Entries with Decimal Positions

When the entries of a table are numeric, it is necessary to specify in column 44 the number of decimal positions (0-9) in each entry. Even if a numeric entry contains no decimal positions, a 0 must still be entered to indicate numeric data. When decimal positions are not specified (column 44 left blank, as in Figure 8-5), RPG II considers the table entries to be alphameric.

Sequence of Table Entries

For TABITM, the item codes need not be in any particular order. Thus, column 45 is left blank in Figure 8-5. However, if the table entries are in ascending or descending order, an *A* or a *D* is entered under Table Sequence (column 45). Note that if a table is to be in sequence the entry is made on the Extension sheet. For input files, other than table files, the sequence entry is always made on the File Description sheet.

Coding the Table Lookup Operation (LOKUP)

Once the table input records have been described, you tell RPG II to search the table by coding the LOKUP operation on the Calculations sheet. This involves specifying:

1. The LOKUP operation code.

2. The name of the table to be searched.

3. The data which is being searched for.

4. Conditions which must be satisfied for a successful search.

As shown in Figure 8-7, the operation code LOKUP is entered in columns 28-32 of the Calculations sheet. This operation code causes a search to be made for a particular item in the table named in Factor 2. Thus the name of the table being searched, TABITM, is entered under Factor 2 (columns 33-42), beginning in column 33. Remember that the table being searched must have been previously described on the Extension sheet.

Factor 1 (columns 18-27) contains the field name of the data which is used for comparison during the table search. In this case, the second field of the sales record (called ITMORD) contains the code for the item ordered (the search data). Thus, ITMORD is entered, beginning in column 18.

The fields on each SALES record have been described on the Input sheet. The field (ITMORD) which contains the search data must have been described so that it agrees with the way the table entries have been described. That is, the search data and the table entries must have the same length, the same number of decimal positions, and the same format (alphameric or numeric). As shown in Figure 8-8, both ITMORD and the table entries are defined as three characters long and as alphameric (no decimal positions specified).

In this case, the item which is being searched for must be an exact match of the search data. This is referred to as searching for an *equal* condition only. The lookup procedure would begin at the first element of TABITM and continue, one element at a time, until an equal entry is found. For this reason, it isn't necessary that the table elements be in any particular order when searching for an equal condition.

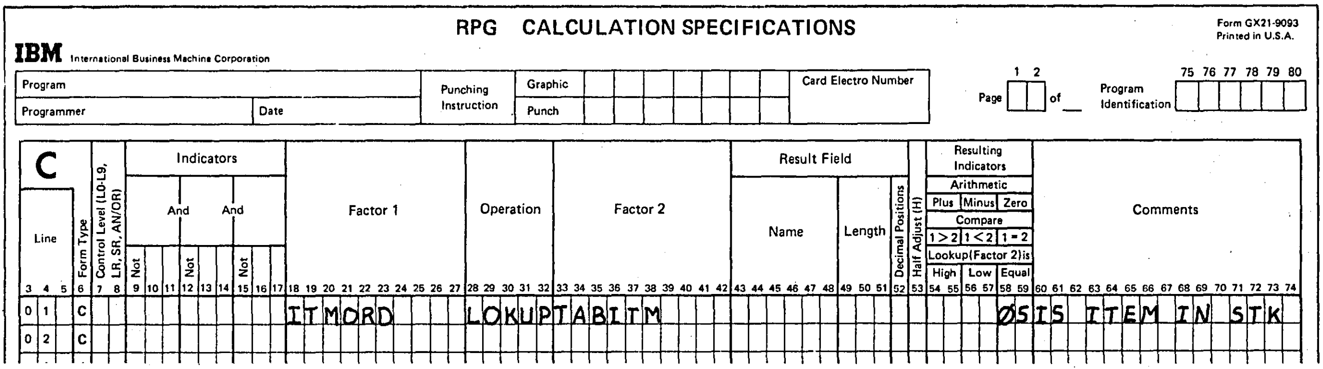
Of course, if the item searched for is not found after checking all elements of TABITM, the company does not carry that item in stock. Thus, the equal condition must be satisfied if this search is to be successful.

But, how will you know if the search was successful? To determine this, you must know if an equal match was found or not. Thus, you should assign a resulting indicator for the lookup which will turn on if (and when) the equal condition is satisifed.

As shown in Figure 8-7, the indicator 05 is assigned by entering 05 on the Calculation sheet under Lookup Equal (columns 58-59). If an entry cannot be found in the table to correspond to the search data, the 05 indicator will turn off.

TWO TABLE SEARCH

As you have seen, a single table can be searched merely to see if certain information (an item code) is in the table. However, usually you will want to do more than this. For example, when the orders are shipped, you will also want to send bills to each customer for the amount of the order. Before RPG II can print the bills, it must calculate the amount each customer owes. To do this, the unit cost of each item must be determined. The unit cost of an item is then multiplied by the number ordered to give the total amount due from a customer for that type of item.

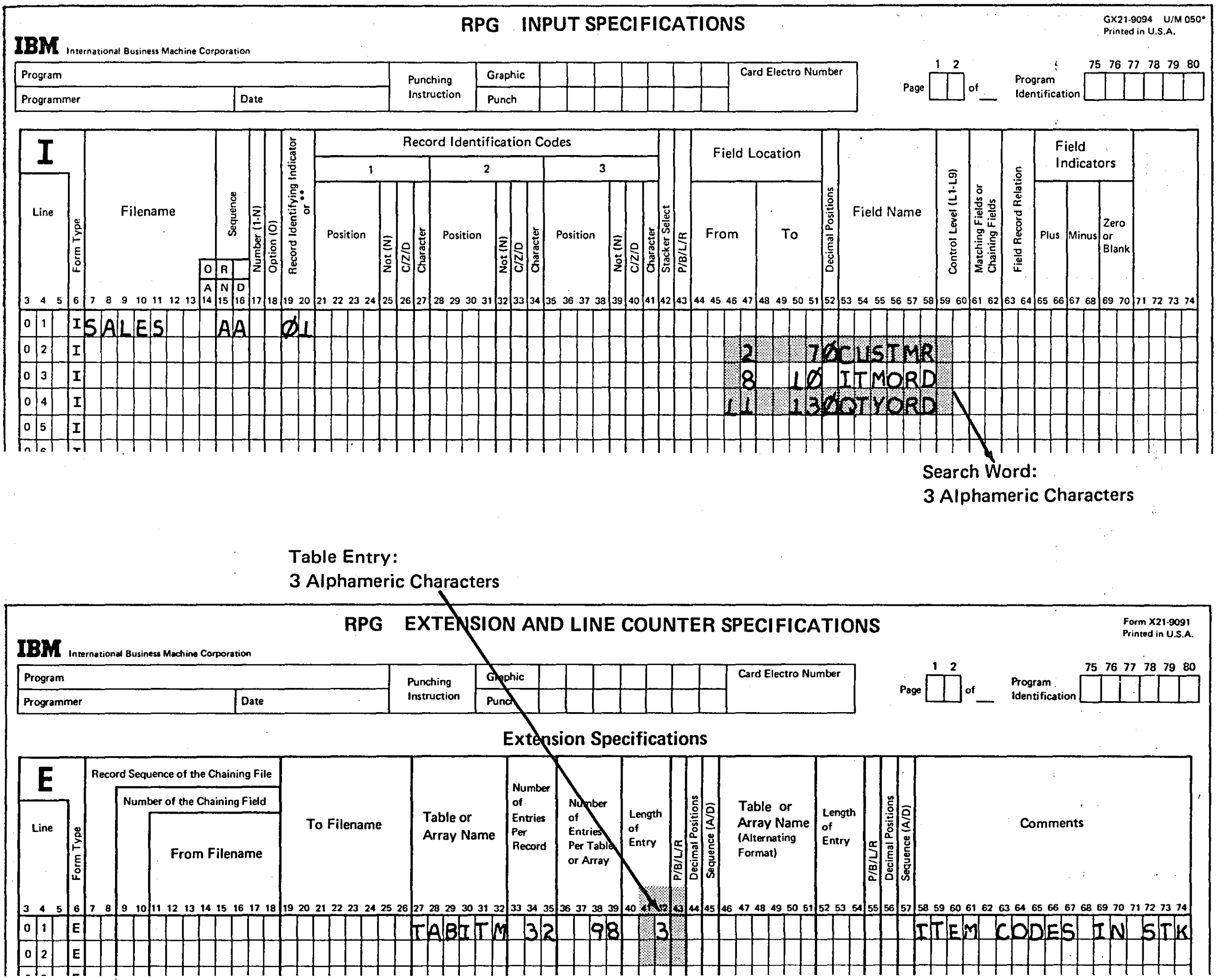


RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation — Form GX21-9093, Printed in U.S.A.

Line	Form Type	Factor 1	Operation	Factor 2	Resulting Indicators (Equal)	Comments
01	C	ITMORD	LOKUP	TABITM	Ø5	IS ITEM IN STK
02	C					

Figure 8-7. Specifying a Single-Table Search



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation — GX21-9094 U/M 050*, Printed in U.S.A.

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Field Location From	Field Location To	Decimal Positions	Field Name
01	I	SALES	AA	Ø1				
02	I				2	7	Ø	CUSTMR
03	I				8	1Ø		ITMORD
04	I				11	13	Ø	QTYORD
05	I							

Search Word:
3 Alphameric Characters

Table Entry:
3 Alphameric Characters

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

IBM International Business Machine Corporation — Form X21-9091, Printed in U.S.A.

Extension Specifications

Line	Form Type	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	Comments
01	E	TABITM	32	98	3	ITEM CODES IN STK
02	E					

Figure 8-8. Specifying the Same Length for Search Word and Table Entry

At this point, you have already created a table (TABITM) listing the codes of all items the company sells. The unit cost for each item (a 5-digit field on each inventory file record) can be stored in a second table, called TABCST. This table should be organized such that the position of a cost within TABCST corresponds to the position of its related item code in TABITM (Figure 8-9).

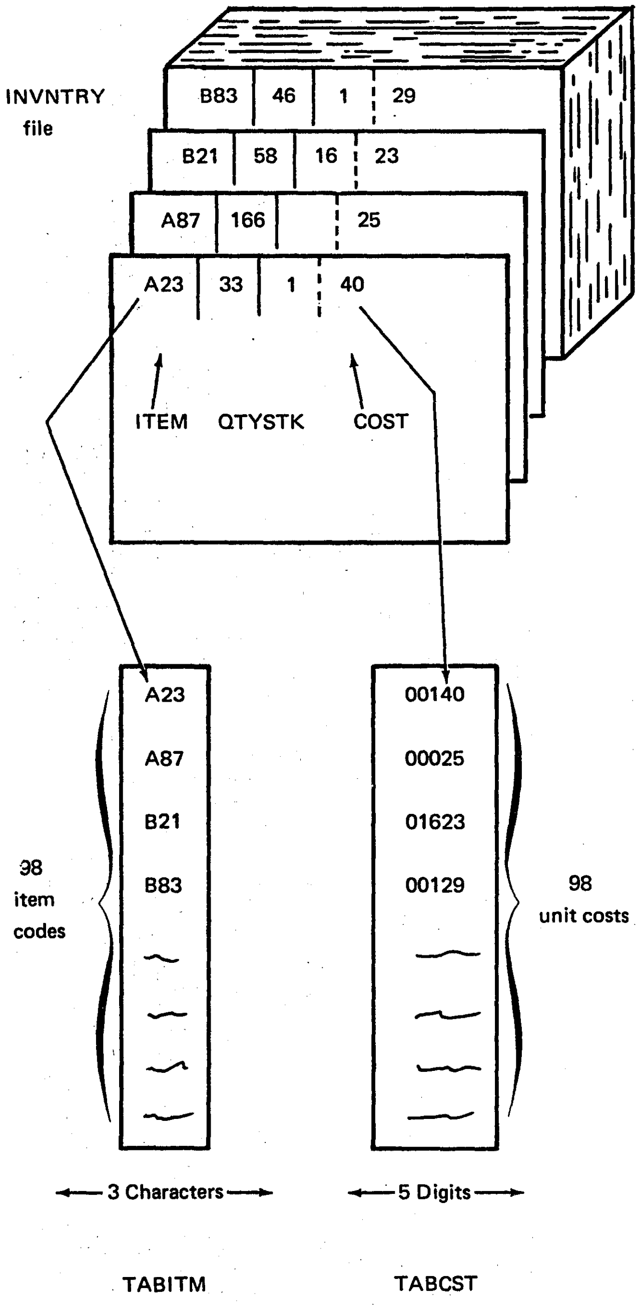


Figure 8-9. Creating Two Related Tables

Let's take a look at one of the records in the SALES file (Figure 8-10) and see how the two tables can be used in determining the appropriate cost for the item ordered. The procedure is much the same as the way you located Ken Adams' telephone number in the telephone directory.

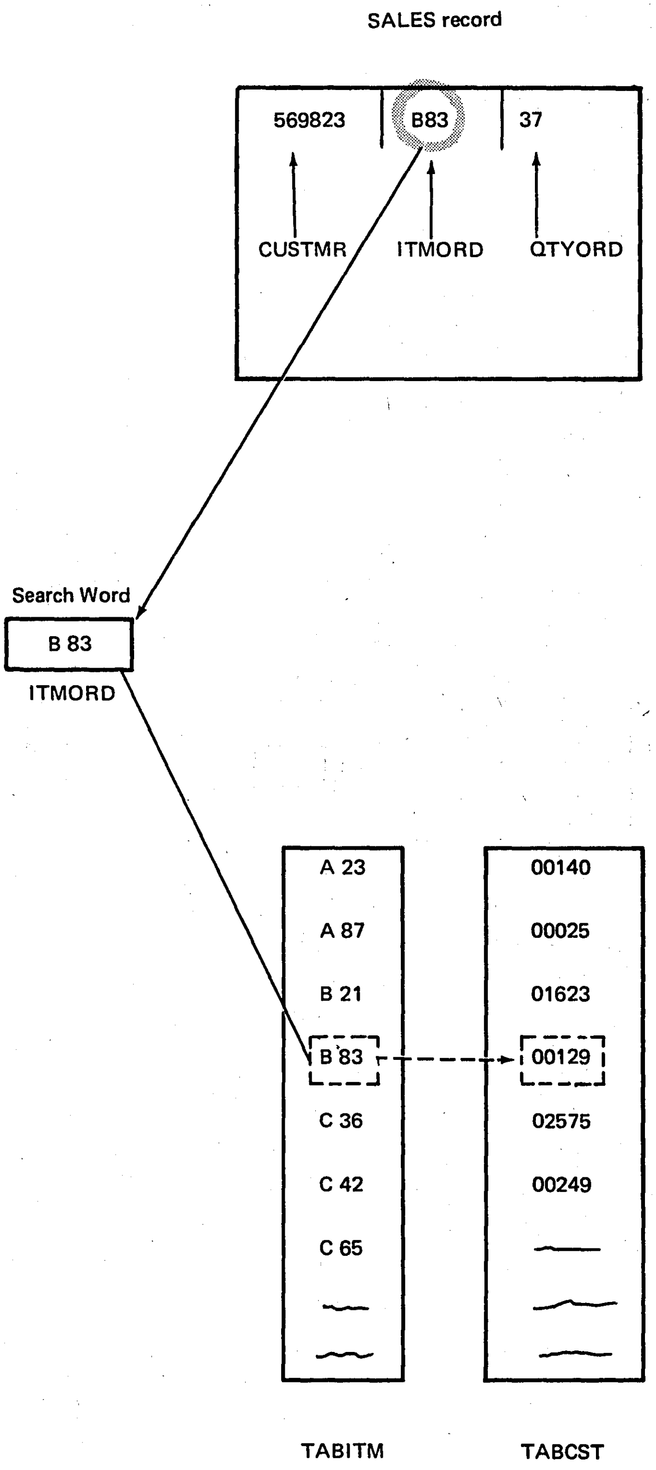


Figure 8-10. Performing a Two-Table Search

First, TABITM is searched to find the table element which contains the same item code as the search word (B83) on the SALES record. If no match is found, that SALES record can be selected into a special stacker so the customers can be notified of the orders which can't be filled. However, in this case, the equal entry is found in the fourth element of TABITM. The entries in TABCST were organized to correspond with entries in TABITM. Therefore, the fourth element of TABCST contains the unit cost for the search word. The program can then use the information referenced (unit cost) to calculate the amount to be billed.

When two related tables are used, as in this example, actually only one table is searched. When the search condition (in this case, an equal match) has been satisfied, the data in the corresponding position or element of the second table (related table) becomes available. Thus, the first table is used as a means of locating data in the second table. In a telephone directory, a person's name is used as a means of locating his telephone number.

Designing Table Input Records for Two Tables

Records With Entries For Only One Table

In designing the table input records for TABITM, you were concerned only with entries for a single table. Thus, four table input records were created to contain only item codes.

Another set of table input records could be created to contain the 98 unit cost entries for TABCST. Each unit cost entry (from the inventory records) requires five columns (three digits for dollars, two digits for cents). Again, it is up to you to determine how many records you wish to use and the number of entries to be contained in each record. For example, let's say that 18 entries (columns 1-90) are punched into each of five table input cards and the remaining eight entries are punched into columns 1-20 of a sixth card.

As shown in Figure 8-11, you would then have two separate table input files: records for TABITM containing only item codes and TABCST records containing only unit cost amounts.

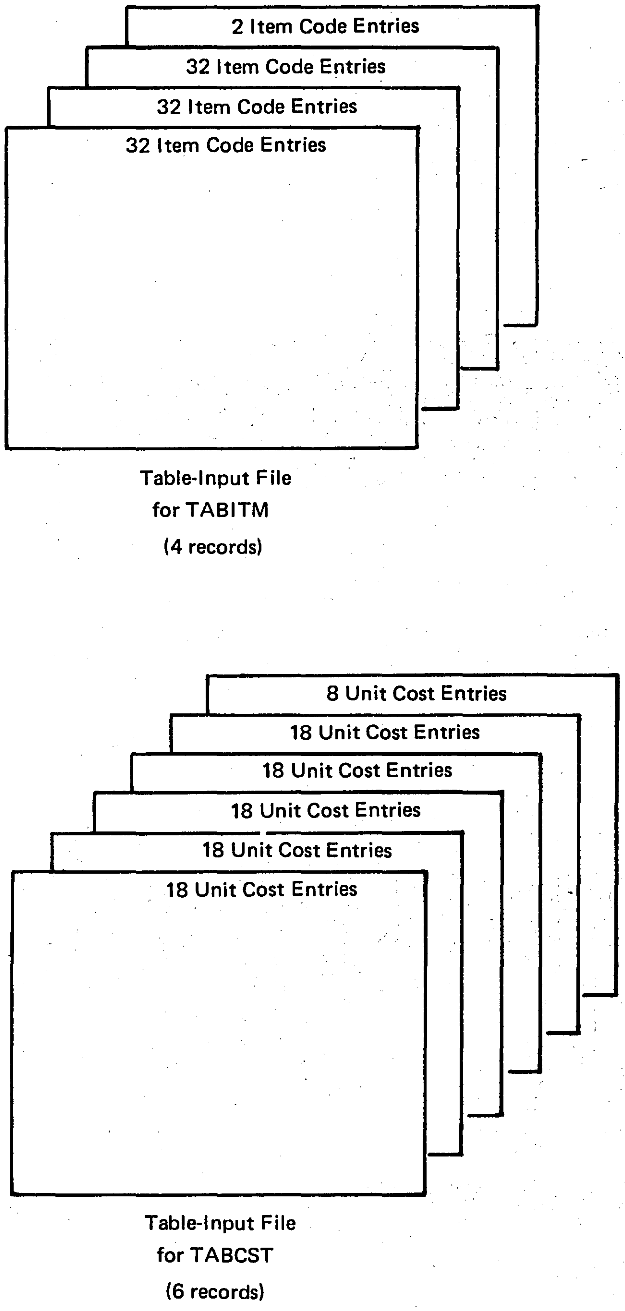


Figure 8-11. Separate Records for Each Table

Records With Entries For Two Tables

Another method of designing table input records allows you to use only one table input file for both tables. This may save input record space and will usually reduce the number of RPG II specifications needed to describe the table input records. Entries from the first table are alternated with entries from the second table (Figure 8-12). The records are then referred to as alternating format table input records.

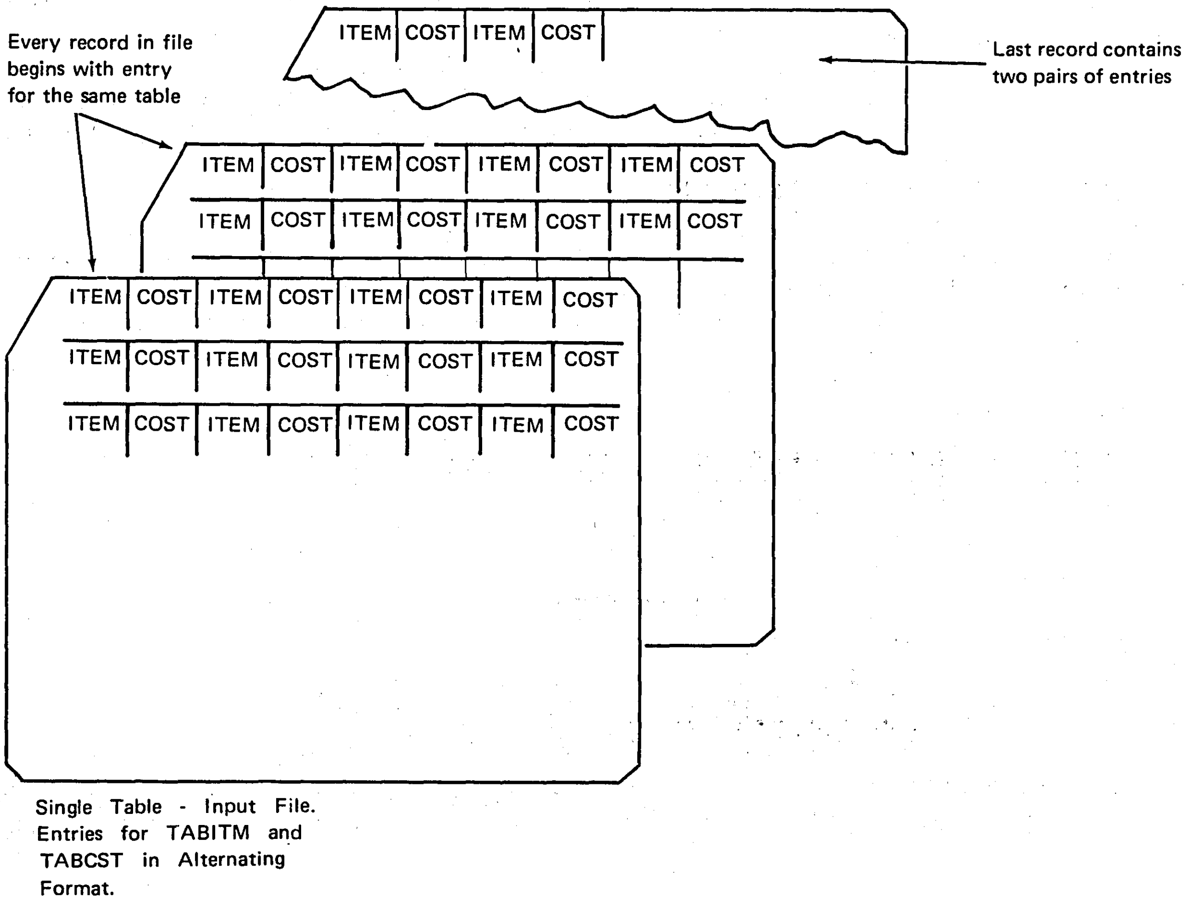
As you can see, a table input record can begin (position 1) with an entry from either the first table or the second table. However, if you decide to use the alternating format, every record in the table input file must begin with the same type of table entry (table 1 or table 2).

The number of entries that you put on an alternating format, table input record is still up to you. If you wish, a single record can contain one code for TABITM and one unit cost from TABCST. Or, the record may contain as many pairs of related entries as possible. In this case, all records, except the last, must contain the same number of entries.

For TABITM and TABCST, each pair of related entries will require eight card columns. By punching 12 pairs of entries in a record, an entire card can be filled. Thus a table input file to contain all entries (98 pairs) from both tables can consist of nine alternating format, table input cards. The first eight records might each contain 12 pairs of entries and the last record might contain two pairs of entries.

As mentioned before, entries for TABITM are each 3-character alphameric data. For TABCST, 5-digit numeric entries are needed. All of the entries for a single table must be alike; that is, all alphameric or all numeric. However, the entries for an alphameric table and the entries for a numeric table can both be on the same table input record when an alternating format is used.

Although each table input record contains entries from both tables, actually two separate tables are created in storage from these records. The RPG II Compiler knows that two tables are to be set up, rather than a single table, because of the way the records are described on the Extension sheet.



Every record in file begins with entry for the same table

Last record contains two pairs of entries

Single Table - Input File. Entries for TABITM and TABCST in Alternating Format.

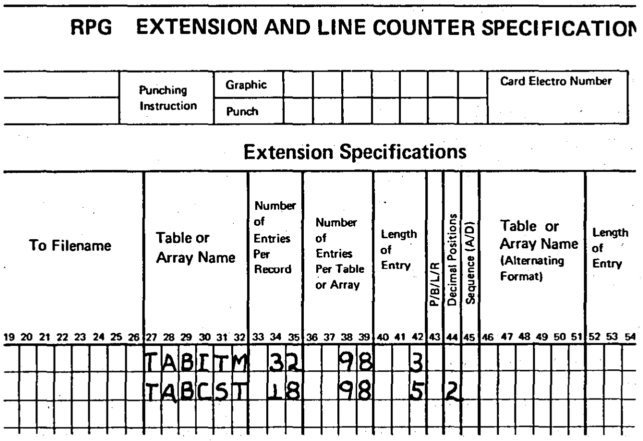
Figure 8-12. Alternating-Format Table-Input Records

Describing Two Tables with Extension Specifications

When two related tables are used in a program, you have the choice of designing two table input files (one for each table) or only one table input file consisting of alternating format table input records.

If you decided to set up TABITM and TABCST using separate files, as discussed earlier, the two tables would be described as shown in Figure 8-13. Since each table has its own set of input records, a separate line of extension specifications is needed for each table.

Now take a look at Figure 8-14 which describes the same two tables. Notice that only one extension line is coded when alternating format, table input records are used. Since one line is coded for each set of input records, the second table of the alternating format record must be entered on the same extension line as the first table.



RPG EXTENSION AND LINE COUNTER SPECIFICATIOI

	Punching Instruction	Graphic		Card Electro Number
		Punch		

Extension Specifications

To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)
	TABITM	32	98	3					
	TABCST	18	98	5		2			

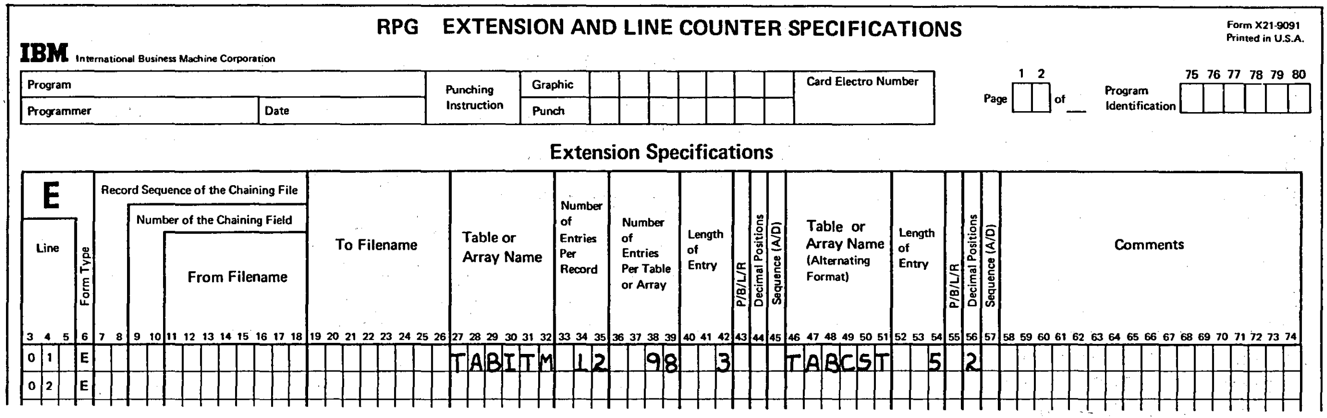
Figure 8-13. Describing Separate Table-Input Records

Remember that two separate tables are created, even if you use alternating format records. Therefore, both tables must have unique names, which are specified on the Extension sheet. The table whose entry appears first on a table input record is named in columns 27-32. The name of the alternating table (in this case, TABCST) is entered in columns 46-51 *of the same line.* The alternating table is always the table whose entry is the second one in a pair of related entries.

Notice that Number of Entries Per Record (columns 33-35) and Number of Entries Per Table or Array (columns 36-39) do not have corresponding specification columns for alternating tables. Since the number of entries per record and number of entries per table must be the same for each of the alternating tables, separate specifications are not needed for the second table.

However, the Length of Table Entry (columns 52-54) must be indicated since it may be different for the two tables. In Figure 8-14, a *3* has been entered in columns 40-42 as the table entry length for TABITM, while a *5* has been entered in columns 52-54 specifying the length of the unit cost entries for TABCST.

The unit cost entries for TABCST are in the form of 12467, which would represent $124.67. Therefore, a *2* has been entered for the number of decimal positions (column 56). This specification indicates that the entry is numeric and contains two decimal positions.



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program		Punching Instruction	Graphic		Card Electro Number	Page __ of __	Program Identification (75-80)
Programmer	Date		Punch				

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field / From Filename (9-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E				TABITM	12	98	3				TABCST	5		2		
02	E																

Figure 8-14. Describing Alternating-Format Table-Input Records

The table entries for TABCST are not in any special alphabetic or numeric order which the RPG II Compiler program could check for accuracy. Thus, no table sequence is specified for TABCST (column 57 is left blank). Likewise, telephone numbers in a directory are not in sequence. However, just as phone numbers are arranged to correspond with related names, the TABCST entries have been arranged so a particular unit cost corresponds with its related item code.

Coding the Table Lookup Operation (LOKUP)

Now that both tables have been set up, TABITM is to be searched to find the table element which contains the same item code as the code (search word) on the SALES record. This simple lookup procedure is coded with the same calculation specifications as you used before (Figure 8-15). The field name or literal constant which contains the search code is entered under Factor 1 (ITMORD), the LOKUP operation is specified in columns 28-32, and the name of the table being searched (TABITM) is entered under Factor 2. If an equal match is found in searching the table, resulting indicator 05 will be turned on, as specified in columns 58-59.

However, you want to do more than just see if the item ordered is carried in stock. If the search is successful, you also want to determine the appropriate unit cost for the ordered item.

To reference the second table (TABCST), additional specifications must be entered on the same line of the Calculation sheet. As Figure 8-15 shows, the name of the table (TABCST) from which you wish data to be made available is entered as the Result Field (columns 43-48). If (and when) the equal search is satisfied, the corresponding data *looked-up* in TABCST will be made available.

Field length and decimal positions have not been entered in columns 49-52. These specifications are not required since the table named under Result Field (TABCST) was previously described on the Extension sheet. However, if you want to enter these specifications again on the Calculation sheet, the numbers must agree with those in the extension specifications. For TABCST, field length must be *5* (in column 51), the length defined for a TABCST entry, and the number of decimal positions must be *2* to agree with the Extension sheet.

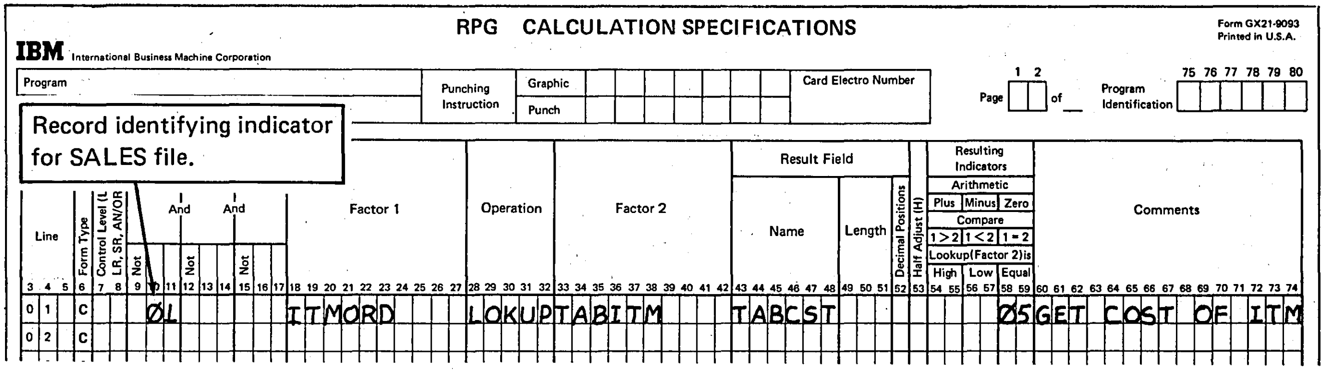
USING TABLE DATA IN CALCULATIONS AND OUTPUT

You perform a table lookup because you want the information referenced to be used in some way. In some programs, you may wish to use the data in a calculation. At other times, you merely want the data to be used for output.

Conditioning Operations on the Basis of a Table Lookup

In the sales job, the specifications for a LOKUP operation instructed the program to search TABITM for an entry equal to an item in a SALES record and then make available the appropriate unit cost entry from a second table. If the search was successful, resulting indicator 05 would have been turned on.

Providing the item code is carried in stock (successful search), you want to use its unit cost to determine the amount the customer owes for the number of items ordered. In other words, the calculation specification in line 02 of Figure 8-16 should be performed.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

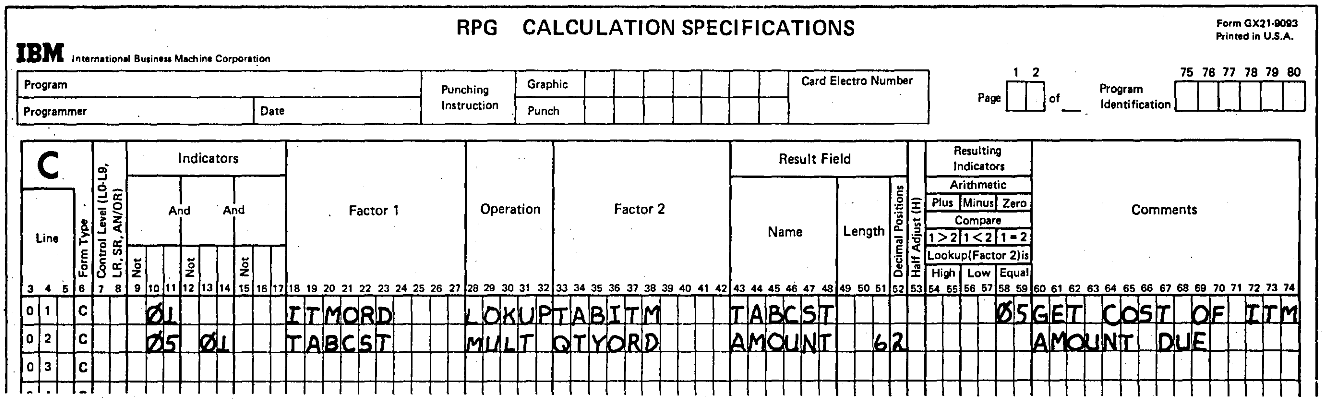
Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction: Graphic, Punch | Card Electro Number | Page (1 2) of | Program Identification (75-80)

Record identifying indicator for SALES file.

Line (3-5)	Form Type (6)	Control Level (L), LR, SR, AN/OR (7-8)	Indicators: Not, And, Not, And, Not (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators Arithmetic Plus, Compare 1>2, Lookup (Factor 2) is High (54-55)	Minus, 1<2, Low (56-57)	Zero, 1=2, Equal (58-59)	Comments (60-74)
01	C		ØL	ITMORD	LOKUP	TABITM	TABCST						Ø5	GET COST OF ITM
02	C													

Figure 8-15. Specifying a Two-Table Search



RPG CALCULATION SPECIFICATIONS

Line	Form Type	Indicators	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Resulting Indicators (Equal)	Comments
01	C	Ø1		ITMORD	LOKUP	TABITM	TABCST			Ø5	GET COST OF ITM
02	C	Ø5	Ø1	TABCST	MULT	QTYORD	AMOUNT	6	2		AMOUNT DUE
03	C										

Figure 8-16. Performing Calculations Only After a Successful Search,

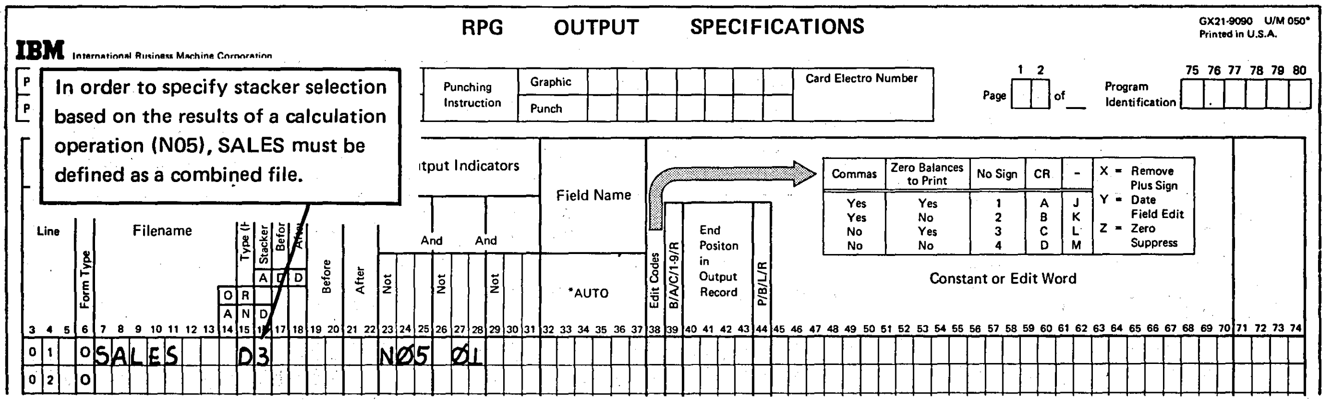
However, if the item code was not found, indicator 05 would have been turned off and no cost entry would be available for the item ordered. For the example given, you could have the SALES record for the current lookup selected into a separate stacker. In this way, you can identify which customers must be notified that their orders cannot be filled.

Note: In order to select a stacker for input records on the basis of a calculation operation (for example, LOKUP), the file must be defined as a *combined* file on the File Description sheet and the stacker must be specified on the Output-Format sheet. See the chapter entitled *Card Output Operations* for a discussion of stacker selection.

Ordinarily, all calculation specifications are performed before any output is done. But if the search for an item was unsuccessful, you would be unable to calculate a bill for the customer correctly.

Therefore, there must be a way to bypass the calculations if the lookup is unsuccessful. This is done by entering 05 as a conditioning indicator in columns 10-11 (see Figure 8-16). The calculation operations will be done only when 05 is on. In this way, the same resulting indicator used to determine the results of the lookup becomes a conditioning indicator to determine if a calculation operation should be performed.

If the item is not carried in stock (05 off), any calculations conditioned by 05 are not performed. The program then performs the output specifications since there are no more calculations to be done. (See Figure 8-17.) For this example, a card is selected into a different stacker only when the search is unsuccessful, not for every item ordered. Thus, the output specification is conditioned by entering *N05* (05 *not* on) and the record identifying indicator in columns 23-28.



RPG OUTPUT SPECIFICATIONS

In order to specify stacker selection based on the results of a calculation operation (N05), SALES must be defined as a combined file.

Line	Form Type	Filename	Type	Stacker Select	Output Indicators	Output Indicators
01	O	SALES	D	3	NØ5	Ø1
02	O					

Figure 8-17. Performing Output Only if Search Was Unsuccessful

At this point, then, you know that you may or may not want certain calculations and output specifications performed, depending on the results of a table lookup. This is accomplished by using conditioning indicators.

Referencing Data Following a Successful Search

According to the RPG II program cycle, input specifications are performed first, followed by calculations and then output. Thus, after reading a data record, all calculations for that record and then all output operations for that record are performed, before the next data record is read and processed.

With this logic in mind, let's take a look at the output wanted from the table lookup program just discussed. TABITM contains the codes of all items carried in stock. The related table, TABCST, contains the unit cost for each item. A SALES file contains the records of customer orders, providing the customer number, code for the item ordered, and the quantity ordered.

The purpose of the program is to calculate the amount each customer owes and print the information on a report. The report is to contain more than just the total amount due, however. As shown in Figure 8-18, each order should have a line printed with data from the SALES record (item ordered, quantity ordered), data looked up from TABCST (unit cost), and data from calculations (total amount due).

After the first SALES record is read into the computer, Figure 8-19 shows that TABITM is searched to find an entry equal to the item code on that SALES record (ITMORD). If a successful search is made, indicator 05 is turned on and the table entry looked up is available for use in further calculations or output operations. In this program, you want to multiply the *looked-up* cost by the number of items the customer ordered (field QTYORD on the SALES record).

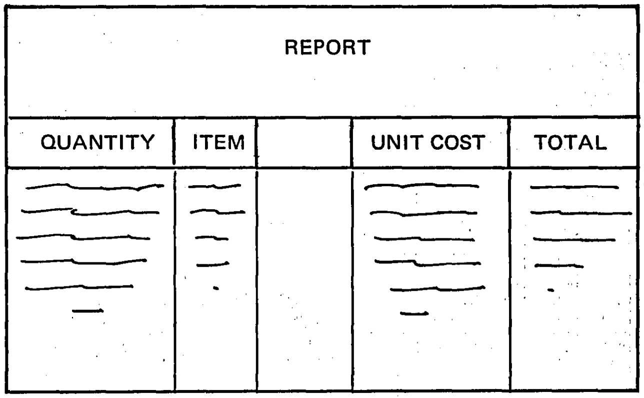
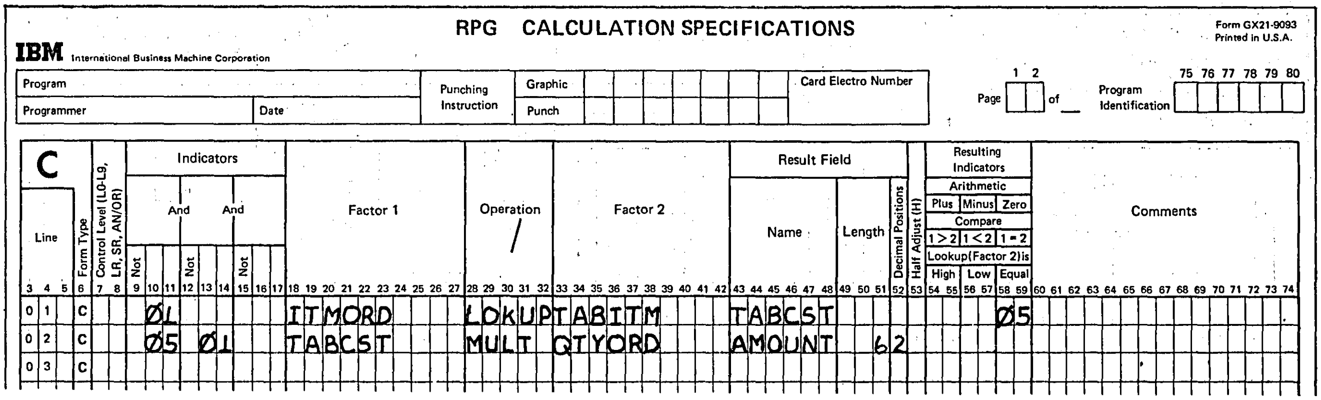


Figure 8-18. Output from a Table Lookup

What is the name of the field containing the cost? The specifications in line 02 of Figure 8-19 shows this instruction. The name of the table (TABCST) which provided the cost has been entered under Factor 1. Whenever a table name is used as a factor in any operation other than a LOKUP (in this case, the operation MULT), the name refers to the data item made available by a LOKUP. Thus, TABCST in line 02 refers to the unit cost of the item just looked up.

Since the table name, TABCST, is used only as a factor and not as the result field of the MULT operation (line 02), the contents of the TABCST data item are not changed. It still contains the unit cost for the item on the sales record.

Once all calculations have been performed, the RPG II program performs the output specifications (Figure 8-20) for this same SALES record. The specifications in line 01 are not performed, since the SALES record is selected into stacker 3 only if the search is unsuccessful (indicator 05 off). However, the rest of the output specifications are performed. Suppose the REPORT file is defined on the File



IBM International Business Machine Corporation

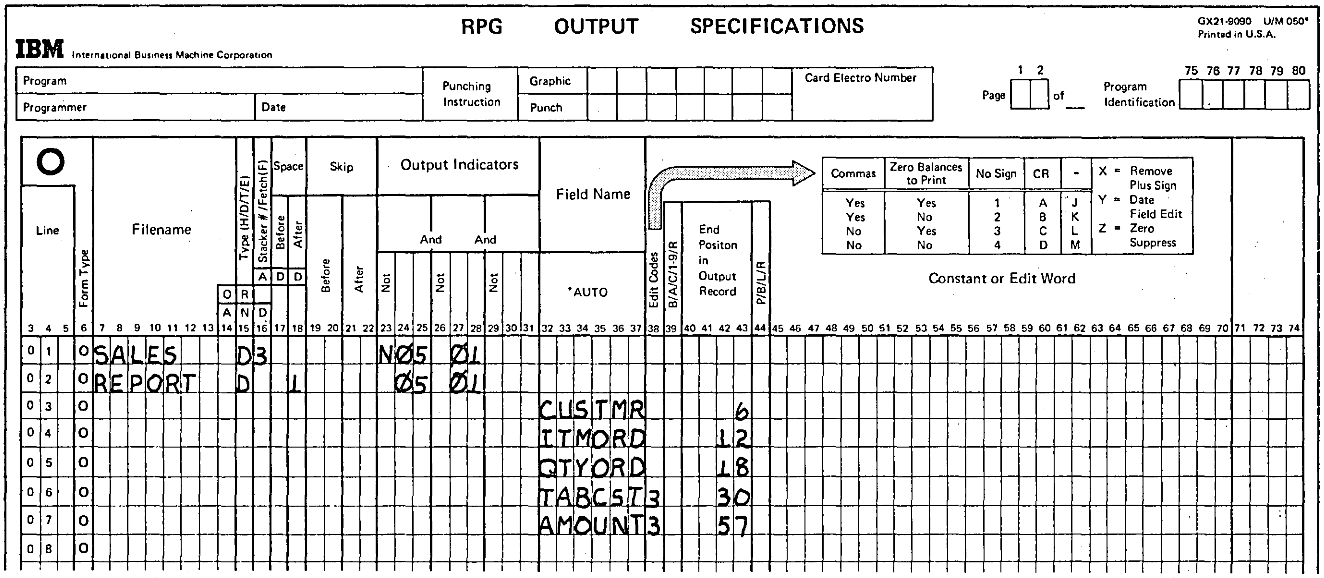
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date
Punching Instruction: Graphic, Punch
Card Electro Number
Page of
Program Identification

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Resulting Indicators High	Low	Equal	Comments
01	C	Ø1	ITMORD	LOKUP	TABITM	TABCST					Ø5	
02	C	Ø5 Ø1	TABCST	MULT	QTYORD	AMOUNT	6	2				
03	C											

Figure 8-19. Referencing Looked-up Table Data



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

```
01 OSALES   D3      NØ5 Ø1
02 OREPORT  D  1     Ø5 Ø1
03 O                         CUSTMR     6
04 O                         ITMORD    12
05 O                         QTYORD    18
06 O                         TABCST3   30
07 O                         AMOUNT3   57
08 O
```

Figure 8-20. Output of Data Looked-up in a Table Search

Description sheet as a printer output file. Lines 02-05 cause the data from the SALES record to be printed on the report. By referencing the name of the table looked up (TABCST) in line 06, the program prints the data contained in the table element, the unit cost for this first SALES record. Line 07 then tells the program to print the amount due, which was previously calculated.

Once all output has been performed for the first record, the RPG II cycle causes the input specifications to be performed again. Thus, the next SALES record is read in, and a table LOKUP is performed for the second record. On a successful search, the unit cost for the second SALES record is made available. The cost for the second item can then be used in calculations and output specifications by referring to the name of the table looked up.

Each table LOKUP operation performed searches for only one entry from a table. This data is then available to be used in calculations and output specifications for that record. The data available does not change until the next table lookup is performed for that table.

TABAMT	TABTAX	TABAMT	TABTAX	TABAMT	TABTAX	TABAMT	TABTAX	TABAMT	TABTAX	TABAMT	TABTAX
.16	.00	16.83	.50	33.49	1.00	50.16	1.50	66.83	2.00	83.49	2.50
.49	.01	17.16	.51	33.83	1.01	50.49	1.51	67.16	2.01	83.83	2.51
.83	.02	17.49	.52	34.16	1.02	50.83	1.52	67.49	2.02	84.16	2.52
1.16	.03	17.83	.53	34.49	1.03	51.16	1.53	67.83	2.03	84.49	2.53
1.49	.04	18.16	.54	34.83	1.04	51.49	1.54	68.16	2.04	84.83	2.54
1.83	.05	18.49	.55	35.16	1.05	51.83	1.55	68.49	2.05	85.16	2.55
2.16	.06	18.83	.56	35.49	1.06	52.16	1.56	68.83	2.06	85.49	2.56
2.49	.07	19.16	.57	35.83	1.07	52.49	1.57	69.16	2.07	86.83	2.57
2.83	.08	19.49	.58	36.16	1.08	52.83	1.58	69.49	2.08	87.16	2.58
3.16	.09	19.83	.59	36.49	1.09	53.16	1.59	69.83	2.09	87.49	2.59
3.49	.10	20.16	.60	36.83	1.10	53.49	1.60	70.16	2.10	87.83	2.60
3.83	.11	20.49	.61	37.16	1.11	53.83	1.61	70.49	2.11	88.16	2.61
4.16	.12	20.83	.62	37.49	1.12	54.16	1.62	70.83	2.12	88.49	2.62
4.49	.13	21.16	.63	37.83	1.13	54.49	1.63	71.16	2.13	88.83	2.63
4.83	.14	21.49	.64	38.16	1.14	54.83	1.64	71.49	2.14	89.16	2.64
5.16	.15	21.83	.65	38.49	1.15	55.16	1.65	71.83	2.15	89.49	2.65
5.49	.16	22.16	.66	38.83	1.16	55.49	1.66	72.16	2.16	89.83	2.66
5.83	.17	22.49	.67	39.16	1.17	55.83	1.67	72.49	2.17	90.16	2.67
6.16	.18	22.83	.68	39.49	1.18	56.16	1.68	72.83	2.18	90.49	2.68
6.49	.19	23.16	.69	39.83	1.19	56.49	1.69	73.16	2.19	90.83	2.69
6.83	.20	23.49	.70	40.16	1.20	56.83	1.70	73.49	2.20	91.16	2.70
7.16	.21	23.83	.71	40.49	1.21	57.16	1.71	73.83	2.21	91.49	2.71
7.49	.22	24.16	.72	40.83	1.22	57.49	1.72	74.16	2.22	91.83	2.72
7.83	.23	24.49	.73	41.16	1.23	57.83	1.73	74.49	2.23	92.16	2.73
8.16	.24	24.83	.74	41.49	1.24	58.16	1.74	74.83	2.24	92.49	2.74
8.49	.25	25.16	.75	41.83	1.25	58.49	1.75	75.16	2.25	92.83	2.75
8.83	.26	25.49	.76	42.16	1.26	58.83	1.76	75.49	2.26	93.16	2.76
9.16	.27	25.83	.77	42.49	1.27	59.16	1.77	75.83	2.27	93.49	2.77
9.49	.28	26.16	.78	42.83	1.28	59.49	1.78	76.16	2.28	93.83	2.78
9.83	.29	26.49	.79	43.16	1.29	59.83	1.79	76.49	2.29	94.16	2.79
10.16	.30	26.83	.80	43.49	1.30	60.16	1.80	76.83	2.30	94.49	2.80
10.49	.31	27.16	.81	43.83	1.31	60.49	1.81	77.16	2.31	94.83	2.81
10.83	.32	27.49	.82	44.16	1.32	60.83	1.82	77.49	2.32	95.16	2.82
11.16	.33	27.83	.83	44.49	1.33	61.16	1.83	77.83	2.33	95.49	2.83
11.49	.34	28.16	.84	44.83	1.34	61.49	1.84	78.16	2.34	94.83	2.84
11.83	.35	28.49	.85	45.16	1.35	61.83	1.85	78.49	2.35	95.16	2.85
12.16	.36	28.83	.86	45.49	1.36	62.16	1.86	78.83	2.36	95.49	2.86
12.49	.37	29.16	.87	45.83	1.37	62.49	1.87	79.16	2.37	95.83	2.87
12.83	.38	29.49	.88	46.16	1.38	62.83	1.88	79.49	2.38	96.16	2.88
13.16	.39	29.83	.89	46.49	1.39	63.16	1.89	79.83	2.39	96.49	2.89
13.49	.40	30.16	.90	46.83	1.40	63.49	1.90	80.16	2.40	96.83	2.90
13.83	.41	30.49	.91	47.16	1.41	63.83	1.91	80.49	2.41	97.16	2.91
14.16	.42	30.83	.92	47.49	1.42	64.16	1.92	80.83	2.42	97.49	2.92
14.49	.43	31.16	.93	47.83	1.43	64.49	1.93	81.16	2.43	97.83	2.93
14.83	.44	31.49	.94	48.16	1.44	64.83	1.94	81.49	2.44	98.16	2.94
15.16	.45	31.83	.95	48.49	1.45	65.16	1.95	81.83	2.45	98.49	2.95
15.49	.46	32.16	.96	48.83	1.46	65.49	1.96	82.16	2.46	98.93	2.96
15.83	.47	32.49	.97	49.16	1.47	65.83	1.97	82.49	2.47	99.16	2.97
16.16	.48	32.83	.98	49.49	1.48	66.16	1.98	82.83	2.48	99.49	2.98
16.49	.49	33.16	.99	49.83	1.49	66.49	1.99	83.16	2.49	99.83	2.99
										100.16	3.00

Figure 8-21. Two Related Tables for Determining Sales Tax

Searching For Low, High, or Equal Conditions

Up to this point, table lookup operations have involved searching for only an equal condition. However, in some cases, you may have a search word which is less than or greater than an entry in the table being searched.

Assume that a 3 percent sales tax is charged in the state in which you do business. Since the tax rate (3 percent) is the same for all amounts, it is a simple data processing operation to calculate the amount of tax for every customer's order. However, merely to show you how a low or high search works, let's use a table lookup to determine the tax due on an order.

The tax due on certain amounts is calculated, and the information is organized into two tables (see Figure 8-21). TABAMT is a list of various amounts of purchases while TABTAX contains the sales tax due on the amounts.

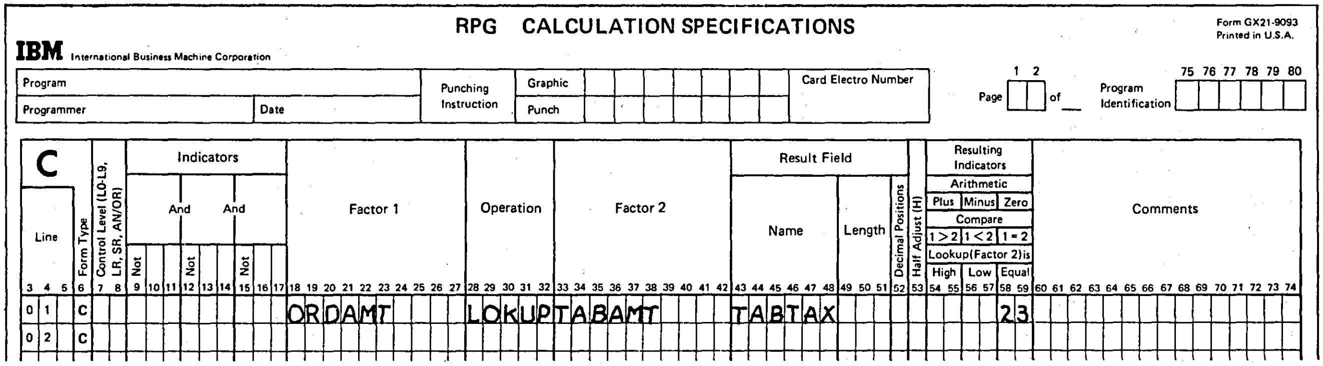
Assume you had to lookup the sales tax for a customer order totaling $9.16. By placing a resulting indicator (01-99) in columns 58-59 of the Calculation sheet (Figure 8-22), you specify that an equal condition is to be satisfied. The computer then searches TABAMT, starting at the beginning of the table, until the table entry *916* (representing $9.16) is located. At that time, the resulting indicator (in this case, 23) is turned on and the sales tax of 27¢ is made available.

On the other hand, what if a customer ordered items which total $5.76? TABAMT contains no such entry. Thus, indicator 23 is turned off, indicating an equal condition cannot be satisfied. What the calculation specifications shown in Figure 8-22, a correct tax amount will never be made available for this purchase.

However, since TABTAX contains all possible tax amounts, a sales tax entry must be present for a purchase of $5.76. Looking at the two tables in Figure 8-21, you can see that a sale of $5.49 requires a tax of 16¢ . But the sale was greater than $5.49; therefore, the tax will be more than 16¢. Take a look at the next entry. For a sale of $5.83, the tax is 17¢. Furthermore, any sale which is less than $5.83, but greater than $5.49 (the previous entry), will also require a tax of 17¢ .

In this case, TABAMT is organized in ascending sequence. Therefore, the TABAMT entry (5.83) which will give the appropriate tax for $5.76 is the first TABAMT entry higher (greater) than the actual search word.

As you learned previously, a resulting indicator must be used to indicate what condition is to be satisfied for a successful search. To specify that a LOKUP is to retrieve a table element higher (greater) than the search word, a resulting indicator must be entered in columns 54-55 (Lookup High) of the calculation specification.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page __ of __
Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators (Not / And Not / And Not)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High (1>2) 54-55	Low (1<2) 56-57	Equal (1=2) 58-59	Comments
01	C			ORDAMT	LOKUP	TABAMT	TABTAX						23	
02	C													

Figure 8-22. Searching for an Equal Condition

Since customer orders can be any amount, the table lookup must be coded to handle all possibilities. For some orders, an equal match can be found in TABAMT; for others, the table entry will be higher than the search word. Therefore, the same resulting indicator (23) should be assigned to turn on if either a high or equal condition is satisfied (see columns 54-55 and 58-59 of Figure 8-23).

A table can also be searched to locate an entry which is lower in value than the search word. In such a case, the table is searched for the entry which is lower (less) than, yet closest in value to, the search word. Searching for a low condition is specified the same way as searching for a high condition, except the resulting indicator is entered in columns 56-57, rather than 54-55.

In coding a table lookup, either one or two conditions may be specified. A particular search may be successful by satisfying:

1. An equal condition only.
2. A high condition only.
3. A low condition only.
4. Either a high or equal condition.
5. Either a low or equal condition.

Searching for either a low or high condition (for the same LOKUP operation) would not be specified, since a majority of items in the table will satisfy one of the two conditions. The condition(s) which must be satisfied can depend on the type of data in the table, the data used as the search word, and the sequence of the data within the table.

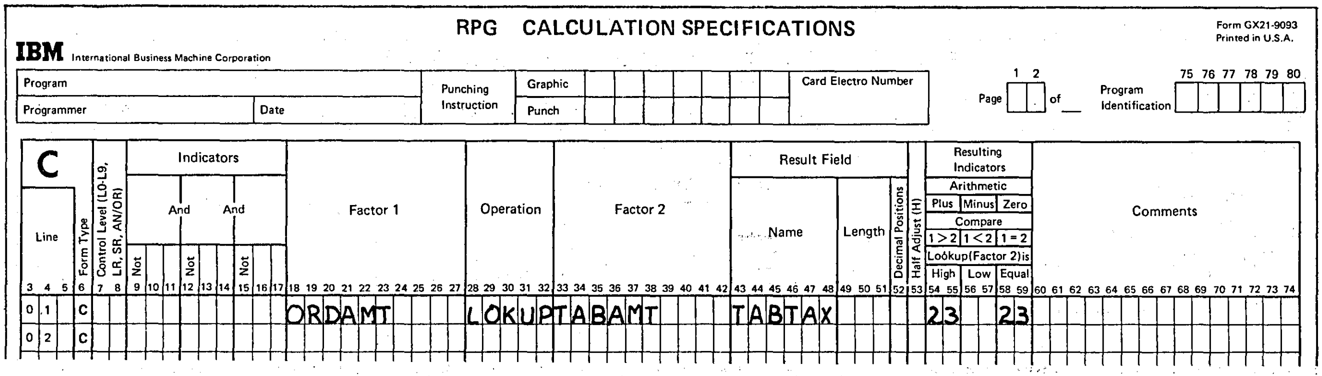
Sequence of Tables

To perform a table search for an equal condition only, it isn't necessary that table entries be in any particular order. Starting at the beginning of the table, the table elements are checked, one at a time, until an equal entry is found or the end of the table is reached, whichever occurs first.

However, when searching for high or low conditions, table entries must be in either ascending or descending sequence. This is because the program must select the entry which is higher or lower than, yet closest in value to, the search word. With the table entries in sequence, the program can determine where in the sequence the search word value would appear if it were in the table. For example, if table elements 2-4-6-9-11 are in ascending sequence, a search word of 7 would naturally come between elements 6 and 9. Thus, element 6 would satisfy the low condition, while element 9 would satisfy the high condition (closest in value and yet higher than the search word).

Likewise, if the table is in descending sequence 11-9-6-4-2, the search word (7) would come between 9 and the 6. Regardless of the sequence (A or D), 9 would satisfy the high condition, while 6 would satisfy the low condition.

If the table elements are not in sequence, as 2-4-11-9-6, the LOKUP might retrieve the wrong element. The elements are checked one at a time from the beginning of the table. Therefore, the computer would determine that a search word of 7 would come between the 4 and the 11. A search for a low condition would incorrectly retrieve element 4, because the next element (11) is greater than the search word (7). While the last element (6) is actually closer in value to the search word, it would not be made available. Likewise, a search for a high condition would retrieve



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

C

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not (9-11)	And: Not (12-14)	And: Not (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field: Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare 1>2 / Lookup(Factor 2)is High (54-55)	Minus / 1<2 / Low (56-57)	Zero / 1=2 / Equal (58-59)	Comments (60-74)
01	C					ORDAMT	LOKUP	TABAMT	TABTAX				23		23	
02	C															

Figure 8-23. Searching for High or Equal Condition

element 11, because it is the first entry encountered after element 4, which is greater than the search word. Element 9, which is closer in value, yet higher than the search word, would not be retrieved since the table is not in sequence.

The sequence of a table is specified by entering an *A* or *D* under Table Sequence (columns 45 and 57) on the Extension sheet (see Figure 8-24). When an entry is made in a sequence column, the RPG II program will check the table entries to ensure they are in the appropriate order (ascending or descending) when loaded.

Generally, table entries are arranged in ascending order, if a sequence is necessary. For certain applications, however, you may find descending sequence more suitable. For example, you may find that the entries with higher values are to be referenced more often. By placing such entries at the beginning of the table (highest to lowest), you may decrease the amount of time required to search a table. Table entries to be in descending order are designated by entering a *D* in the columns 45 and 57, rather than an *A*.

The fact that a table should be in sequence can affect the design of the table input record. In general, when using cards, table input records containing one entry per record (or pair of entries if alternating format is used) are more desirable for sequenced tables. In this way, when a sequenced table is to be updated with additions or deletions the change cards can simply be inserted or removed.

Moving Data in a Table Entry

Suppose you wish to use a table TABCOD to LOKUP data from a related table, TAB123. TABCOD contains codes for all items carried in stock. TAB123 contains information about each of the items.

Up to now, we have discussed table entries containing single items of data. However, a table entry may contain more than one field of information. For example, each entry in TAB123 contains a 15-character item description, followed by a 4-digit unit cost with two decimal positions and a 3-digit quantity in stock. A pair of related entries from TABCOD and TAB123 might appear as in Figure 8-25.

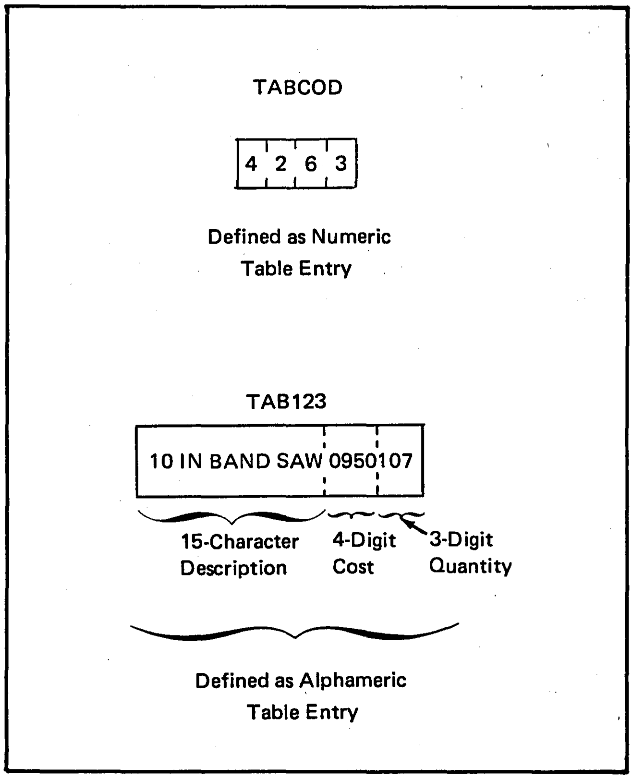
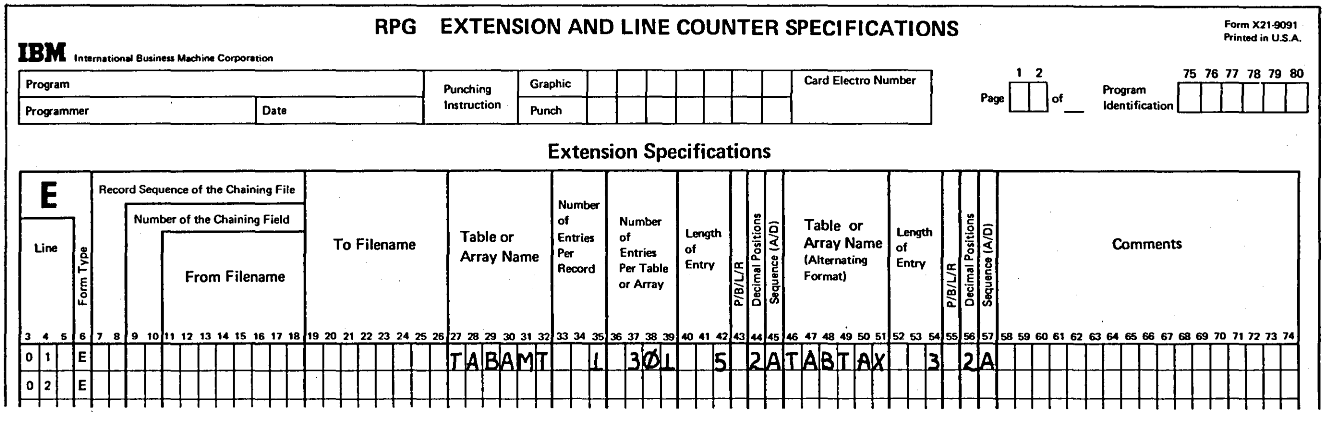


Figure 8-25. Table Entries of More Than One Data Field



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program		Punching Instruction	Graphic		Card Electro Number	Page 1 2 □□ of ___	Program Identification 75 76 77 78 79 80	
Programmer	Date		Punch					

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					TABAMT	1	3Ø1	5		2	A	TABTAX	3		2	A	
02	E																	

Figure 8-24. Specifying Sequence of Table Entries

As you know, following a successful search, you can use the entire looked-up entry in calculations and output merely by specifying the table name. If the table name TAB123 was specified on the Output sheet, the description, cost, and quantity would all be printed or punched, as follows: 10 IN BAND SAW0950107. The data items would be run together, just as they appear in the table entry.

If a table entry contains several fields of data, often you may wish to use only part of the table entry in a particular program. For example, to do your billing, you need only the item description and cost from TAB123. To reference only part of an entry, the data in the TAB123 entry must be separated after the successful search. This is done by moving the data from TAB123 into smaller separate fields, which can then be used in calculations and output.

As shown in line 02 of Figure 8-26, first the contents of TAB123 are moved to the left into a 15-character alphameric field. This isolates the description, which can then be printed by referring to the DESCRP field name on output specifications. To isolate cost, which is in the middle of the table entry, you must first move it, along with one of the outer fields (quantity or description), into a temporary work field. Thus, line 03 moves the rightmost seven characters (cost and quantity) into an alphameric field, WORK. Since the cost is now in the left part of the WORK field, moving the field to the left only four places will isolate cost from quantity. In this way, the field COST can be used to reference only the cost information. Furthermore, COST is specified as a numeric field so that cost data is in the proper format for use in calculations.

The data from the table entry is now separated into new fields which can be used in calculations or to output the single items of data.

Modifying the Contents of a Table

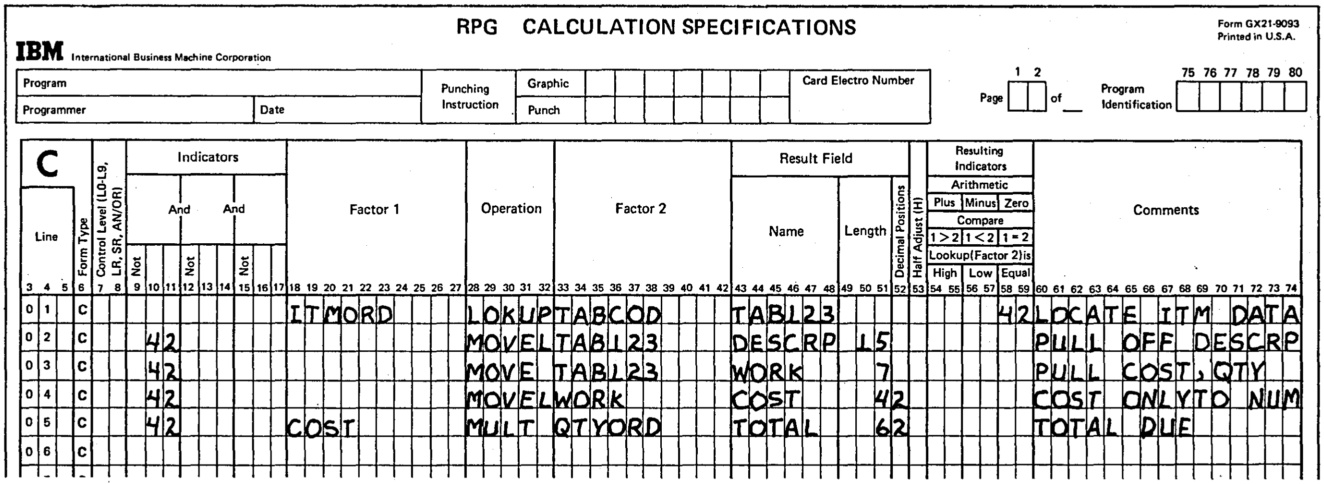
At some point, you may wish to make changes to the data contained in a table. These changes can be temporary, for a particular run; or they can be permanent, such that every time a job is run which references that table, the program uses the changed table data.

Making Temporary Changes to Table Data

Temporary changes to the entries in a table can be made by calculation specifications which are actually a part of your RPG II program. For example, two tables provide the item code (TABITM) and unit price (TABCST) for each part. If you change the price of a part for a particular run, you must necessarily change the entry in TABCST for that part.

As Figure 8-27 shows, TABITM is searched to locate a certain part code. On a successful search, the unit price for that part is made available from the corresponding entry of TABCST. This information is available for the table named under Result Field (line 01).

By using the name of the table referenced (TABCST) in any operation other than the LOKUP (see line 02), you are actually referring to the table entry last looked-up. Thus, by moving the field to TABCST, you are actually storing the new unit price for that item in the table.



IBM International Business Machine Corporation

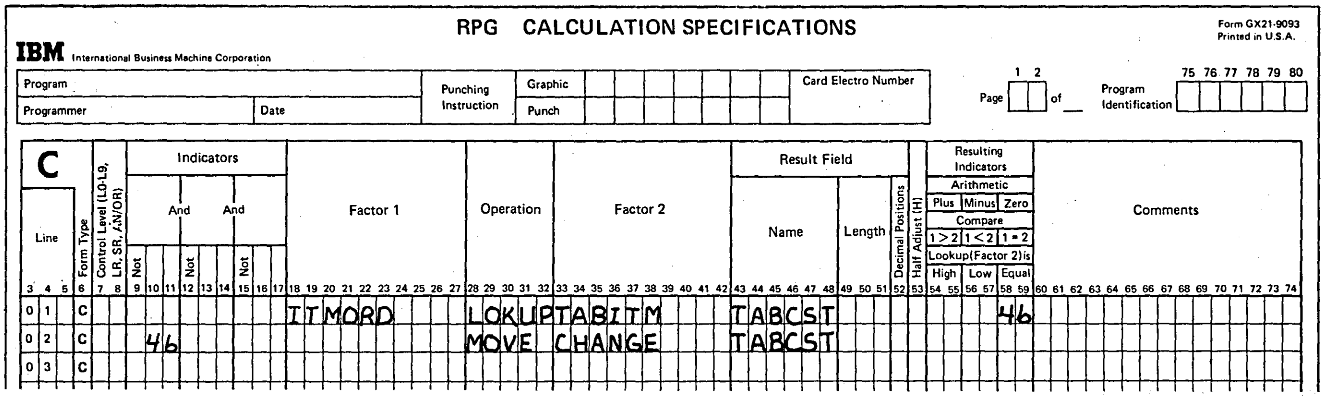
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
01	C			ITMORD	LOKUP	TABCOD	TAB123						42	LOCATE ITM DATA
02	C		42		MOVEL	TAB123	DESCRP	15						PULL OFF DESCRP
03	C		42		MOVE	TAB123	WORK	7						PULL COST, QTY
04	C		42		MOVEL	WORK	COST	4	2					COST ONLYTO NUM
05	C		42	COST	MULT	QTYORD	TOTAL	6	2					TOTAL DUE
06	C													

Figure 8-26. Isolating Part of a Table Entry



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators (Not)	And (Not)	And (Not)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Plus / High / 1>2	Minus / Low / 1<2	Zero / Equal / 1=2	Comments
01	C					ITMORD	LOKUP	TABITM	TABCST						46	
02	C		46				MOVE	CHANGE	TABCST							
03	C															

Figure 8-27. Modifying a Table Temporarily

As you can see, the use of a table name as the result field of a calculation operation is one means of modifying the contents of that table. Since the changes are indicated by specification entries, the changes must be planned for while you are still writing the RPG II program. Otherwise, the instructions could not become a part of the object program.

It is very important to note that any changes made to a table during execution of the program exist for that run only unless additional specifications are made that indicate a permanent change. Thus, the next time the program is run, the original table data is used.

Making Permanent Changes to Table Data

Whether the contents of a table must be permanently changed generally depends on the type of data contained in the table. For example, a table used to keep inventory records will undoubtedly change quite often. Assume a company uses a table to hold the quantity on hand for each part manufactured. Every time the company manufactures more of a particular part or sells (and delivers) a part, the quantity on hand for that part must be increased or decreased, accordingly.

The only way to make a permanent change to the data in a table is to change the table input records. If the data is changed as a result of calculation specifications performed during a run, the changed data can be punched into cards or written to another output device during the run. In this way, the output can be used as table input records for the next run.

Short Tables for Adding New Table Entries

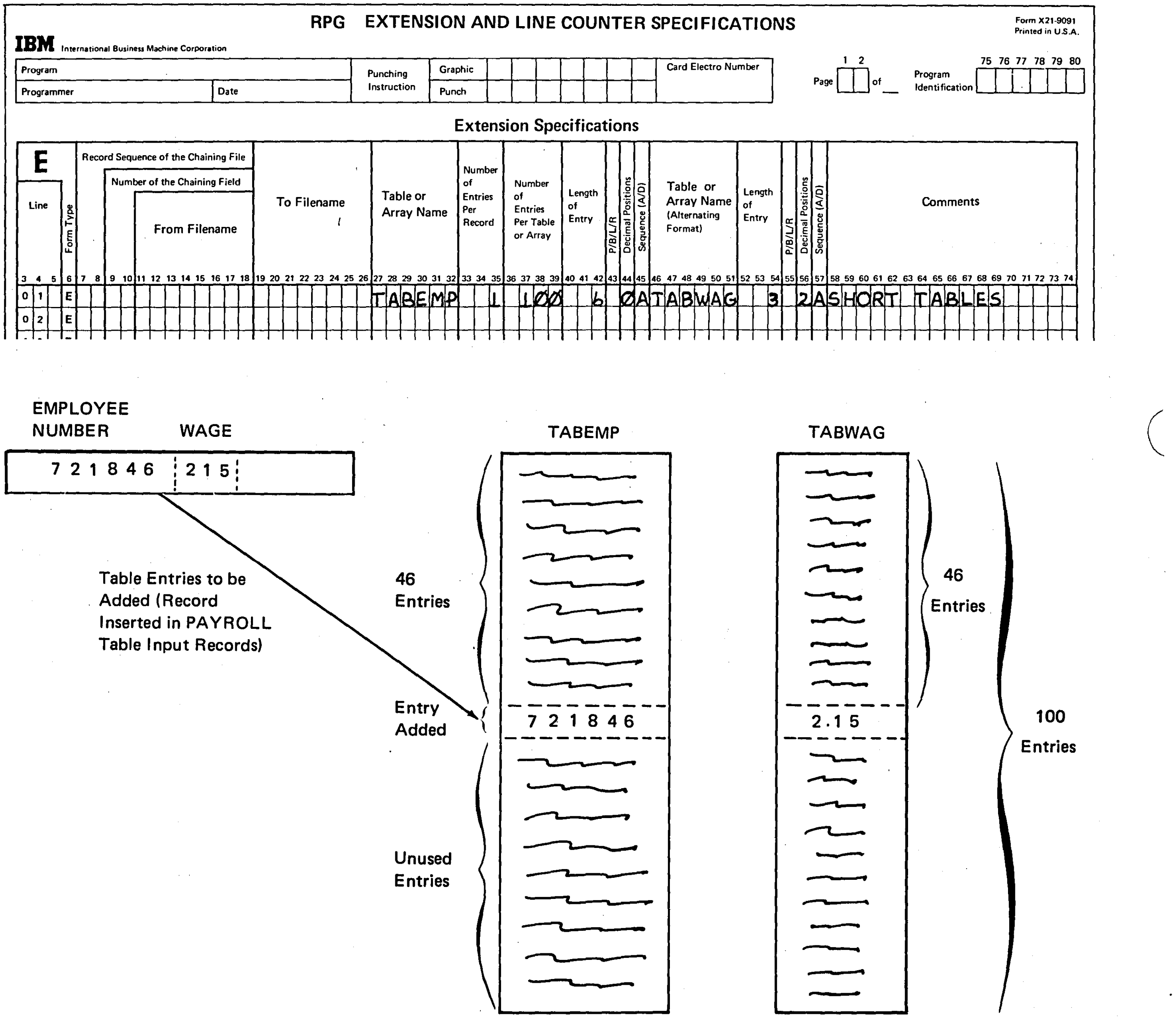
Rather than changing data already stored in a table, there may be cases in which you want to add additional data to an already existing table. For example, assume your company wants to keep a list of employee numbers and hourly wages in two tables, TABEMP and TABWAG. At the present, there are 46 employees on the company payroll; thus, there should be 46 entries in each table. However, you know that the number of employees may increase to about 90-100. Therefore, you will want to add entries to the table as new employees are hired.

At the time the table input records are set up, you must describe them on the Extension sheet. This means that you specify the number of entries per table. If you were to specify only 46 entries per table, it would be necessary to code new extension specifications every time the number of table entries changes. Since the extension specifications are compiled and become part of the object program, it would be necessary to recompile the program to make such a change.

If you know beforehand that the size of your table will increase, you can initially build a short table. In a short table only some of the table entries contain actual table data. The program fills the unused entries with blanks. Thus, TABEMP and TABWAG can be defined as 100 entries each; although for now, you plan to use only 46 entries in each table (see Figure 8-28). Of course, be aware that enough storage will be reserved for 100 entries. The storage required for the unused table entries will not be available to hold any other data.

As new employees are hired, the new table entries can be added by inserting additional table input records. The original extension specifications still correctly describe the table, as they merely indicate the maximum number of entries allowed for each table.

Whether the RPG II source program and the table input records must be recompiled depends on which method was selected originally for loading the table. The methods of loading tables and their effect on short tables is discussed in a following section.



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					TABEMP	1	1ØØ	6		Ø	A	TABWAG	3		2	A	SHORT TABLES
02	E																	

Figure 8-28. Adding Entries to a Short Table

LOADING TABLES

Table data can be loaded into the computer at two different points: at the time your RPG II source program is compiled (compile time tables) or at the beginning of your RPG II object program execution (pre-execution time tables. How often you wish to make permanent changes to the data contained in the table usually dictates the time at which you choose to load that table. The choice should be made as you plan your application, since your decision may affect the design of your table input records and the specification entries required. Furthermore, loading of the table input records differs, according to the type of table used.

Compile Time Tables

Tables loaded at the same time as your RPG II source program are referred to as compile time tables. In other words, the table file is compiled (or translated into the machine or object language) along with the RPG II source program. In this way, the table data is actually a part of the object program. Every time you run the particular object program, then, the table(s) are brought into storage at the same time as the program. As you can see, one definite advantage in creating compile time tables is that you avoid the necessity of loading separate table files into the computer every time you wish to run that object program.

Changing Compile Time Tables

Temporary changes to data in a compile time table exist only for a particular run and are made as easily as for any table. Calculation specifications which have been previously coded in the program can modify any of the table elements.

Making permanent changes to a compile time table or adding new entries to a short compile time table requires recompiling the entire RPG II source program along with the new or changed table input records. The object program produced then contains the current table data. Of course, this process of recompilation requires extra time.

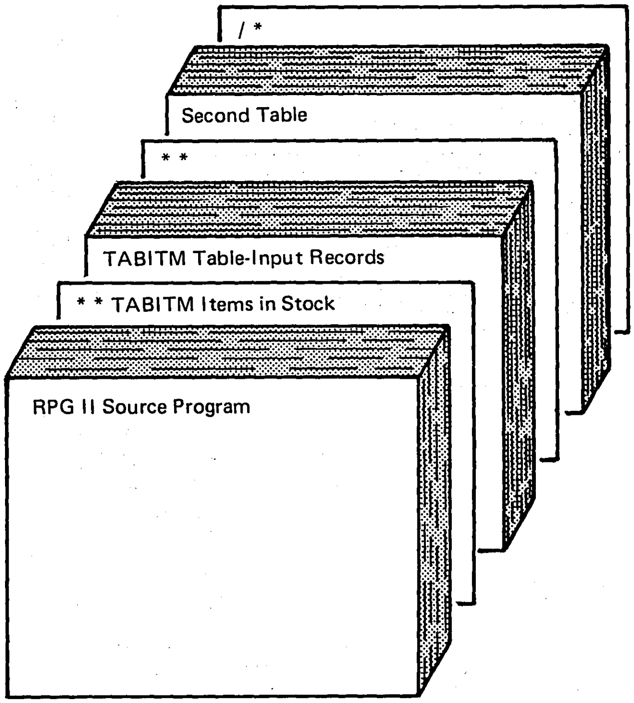
Loading Compile Time Tables

A table to be compiled with your program should follow the RPG II source program (Figure 8-29). There should be a record immediately before the table containing ** in positions 1 and 2. Position 3 must be blank but remaining positions may be used for comments, such as the table name. If more than one table is to be compiled, an ** record should be placed before each table. Furthermore, the compile time tables must be loaded in the same order as they are described on the Extension sheet. The end of file record (/* in positions 1-2) which usually comes at the end of the source program is then placed after the last compile time table.

Model 10 Disk System, Model 6, and Model 15 users may place compile time tables in the source library following the source program. The same record sequence as shown in Figure 8-29 is used. See the applicable reference manuals for your system for specific procedures.

Pre-execution Time Tables

In general, if a table is to be permanently modified often, it is better to create a pre-execution time table. Such a table file is not compiled with your RPG II source program. Instead, only the source program is compiled or translated into the object program. Once the object program has been loaded into the computer to be executed, the table file is loaded separately. Like any other input data file, the table file is then used by the object program, rather than being a part of the program.



Card System Users: The source program and table input records as shown here are placed in the secondary MFCU hopper. The RPG II compiler program is placed in the primary hopper.

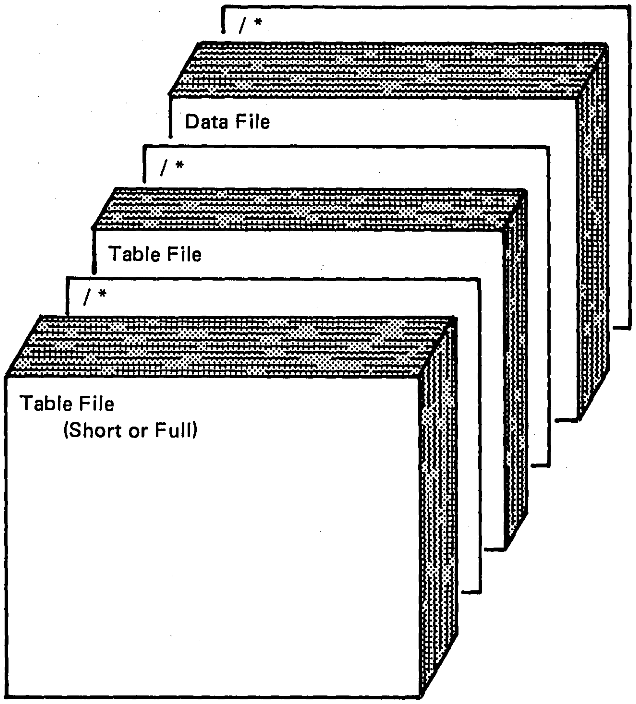
Figure 8-29. Loading Compile Time Tables

Changing a Pre-execution Time Table

Modifying a pre-execution time table takes much less time and effort than changing a compile time table. Modifying the contents of the table permanently (whether a short table or a full table) can be done by inserting and deleting change records. In any event, only the table file is changed. Since there is no need to make changes in the RPG II object program, it isn't necessary to recompile the entire program.

Loading Pre-execution Time Tables

Pre-execution time tables are similar to any other input data files in that the RPG II object program uses the files when the program is executed. However, unlike other data files, pre-execution time tables are read completely before operations involving the tables are done. An end of file record (/*) must follow every pre-execution time table file, regardless of whether the table is short or full (Figure 8-30). The ** record that precedes each compile time table is not used for pre-execution time tables.



Model 10 Card System Users: The table files are loaded from the secondary MFCU hopper. The RPG II object program is loaded from the primary hopper.

Other Systems: Table files loaded at pre-execution time may be loaded from console, cards, disk, or tape.

Figure 8-30. Arrangement of Input for Pre-execution Time Tables

The table files should be in the same order as for compile time tables. All table files are to be loaded in the same order as they are described on the Extension sheet. Furthermore, if both pre-execution time table files and other input data files are to be used by a program, all tables must be loaded before the data files.

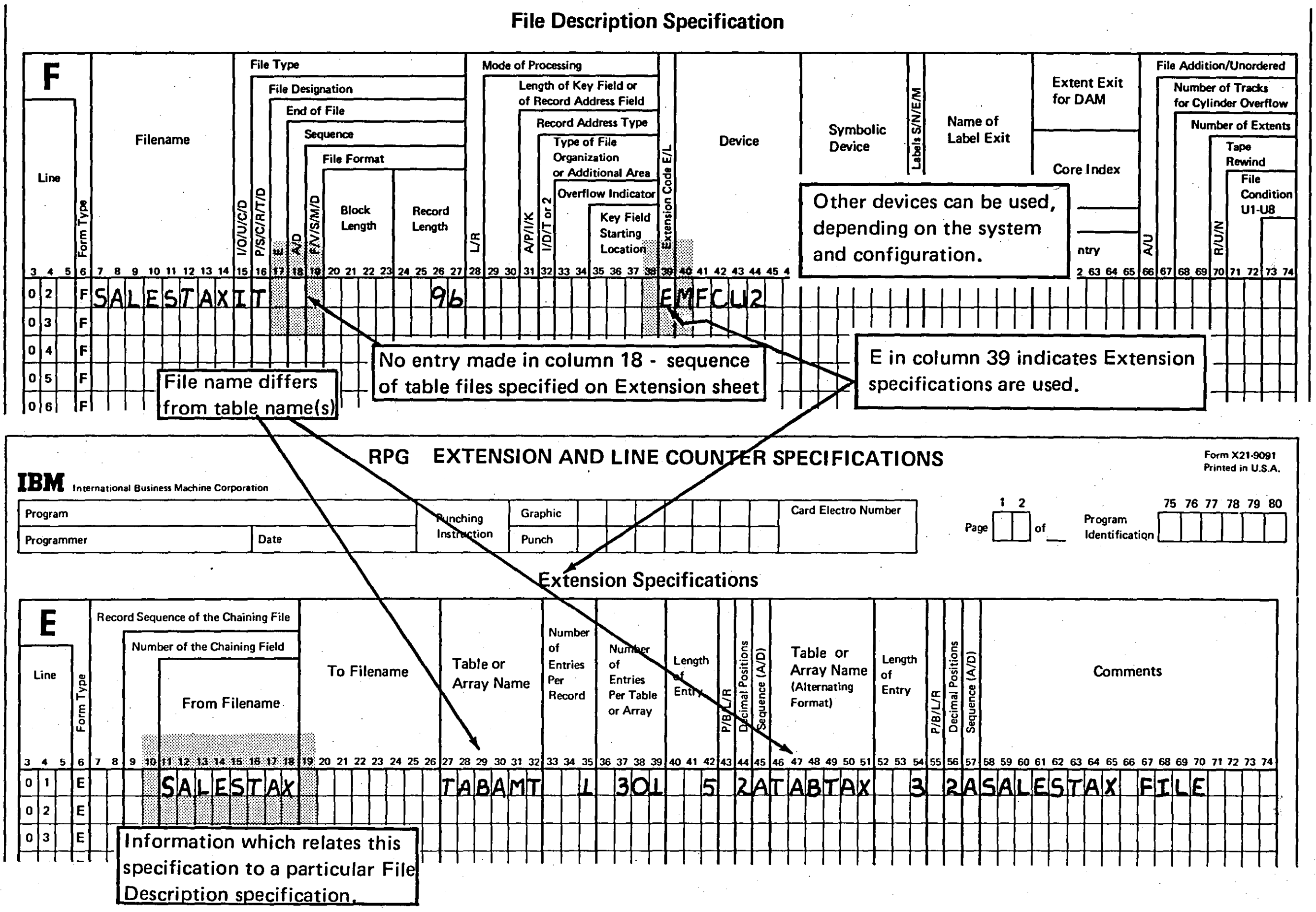
Specifications for Pre-execution Time Tables

Since a pre-execution time table is a separate file to be used by the program, the entire file of table input records must be defined on the File Description sheet, just as any other file must be. This specification sheet is not required for compile time tables because the records are not used as a file. Instead compile time table data becomes a part of the object program.

Figure 8-31 shows the file description specifications required to define a pre-execution time table input file. A filename, different than the table name, should be assigned to the entire table file (columns 7-14). In this case, the file called SALESTAX contains data for both TABAMT and TABTAX. An *I* in column 15 distinguishes that SALESTAX is an input file. Notice, also that the File Designation entry (column 16) must be a *T*, to indicate that this is a table file, as well. The Device entry (columns 40-46) indicates the device from which the table file is read; in this example, we are using MFCU2.

Ordinarily, if an input file is to be in a particular sequence, an entry (A or D) is made in column 18 of the File Description sheet. However, when specifying a sequence for table files, the sequence columns (45 and 57) on the Extension sheet must be used, rather than the sequence column on the File Description sheet. An *E* has been entered in column 39 of the File Description sheet to indicate that the records in this table file are further described on the Extension sheet.

Looking at Figure 8-31, you can see that the filename assigned on the File Description sheet is also entered under From Filename (columns 11-18) of the Extension sheet. This common entry tells the computer that the extension specifications describing TABAMT and TABTAX tables are associated with the SALESTAX file defined on the File Description sheet. For compile time tables, no entry is made in columns 11-18. This is because no file description specifications are made and, thus, no filename is assigned to compile time table records.



File Description Specification

Line	Form Type	Filename	...	Record Length	...	Extension Code E/L	Device
0 2	F	SALESTAXIT		96		E	MFCU2
0 3	F						
0 4	F						
0 5	F						
0 6	F						

Other devices can be used, depending on the system and configuration.

No entry made in column 18 - sequence of table files specified on Extension sheet

E in column 39 indicates Extension specifications are used.

File name differs from table name(s)

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Extension Specifications

Line	Form Type	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
0 1	E	SALESTAX		TABAMT	1	301	5		2	A	TABTAX	3		2	A	SALESTAX FILE
0 2	E															
0 3	E															

Information which relates this specification to a particular File Description specification.

Figure 8-31. Defining a Pre-execution Time Table File

OUTPUT OF AN ENTIRE TABLE

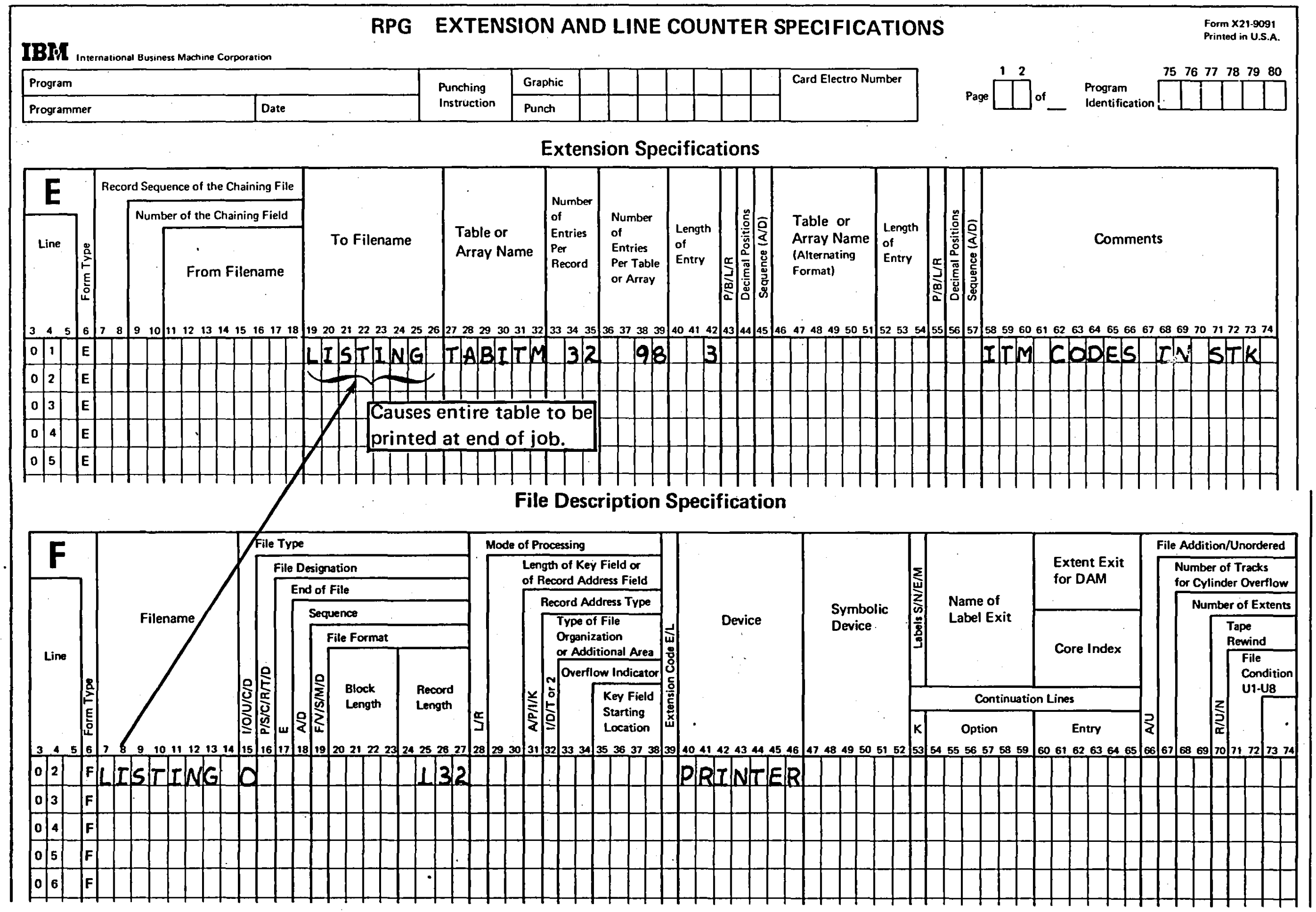
Up to now, we have discussed how to write or punch only individual table entries, one at a time. In this way, only table entries which satisfy a search condition (which have been placed in the hold area) have been used as output.

For various reasons, you may want an entire table written or punched out. Perhaps you want a listing of the table entries to determine if any changes should be made. If your program updates a table, you may wish to output all the entries to be used as table input the next time the program is run.

An entire table can be written or punched out only at the end of the job; that is, after all other output has been completed (LR on). Even if the table is a short table, all entries are put out including those which are unused (blanks or zeros).

Writing or punching an entire table at end of job is very easy to specify. Just as for any type of output, the output file must be given a name and assigned to an output device on the File Description sheet. However, no output specifications are necessary for end of job table output. You merely specify the name of the output file in columns 19-26 of the Extension sheets on the same line as you described the table input records. With the extension specifications shown in Figure 8-32, the table will be put out automatically at end of job.

The output data may not be exactly the same as the input data, because table entries are put out as they are at end of job. Thus, if your program has changed or updated any table entries, the modified data will be put out, not the original data. Except for printer output, however, the format of the table output records will be the same as the input records.



RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

IBM International Business Machine Corporation

Form X21-9091
Printed in U.S.A.

Extension Specifications

Line	Form Type	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	Comments
01	E		LISTING	TABITM	32	98	3	ITM CODES IN STK
02	E							
03	E							
04	E							
05	E							

Causes entire table to be printed at end of job.

File Description Specification

Line	Form Type	Filename	File Type	Record Length	Device
02	F	LISTING	O	132	PRINTER
03	F				
04	F				
05	F				
06	F				

Figure 8-32. Specifications for Output of Entire Table at End of Job

Review 8

(Answers to review questions follow the review.)

1. a. Design the table input records for an inventory table of item numbers. Each item number is a five-digit number. There are 123 unique items in stock. Use either one record per item or one record for a number of items. State the maximum number of records necessary to contain the table data. State the minimum number of records necessary to contain the table data.
   b. Assign a name to the inventory table.
   c. Define the table by coding the necessary extension specifications.
   d. Code the calculation specifications necessary to search the table for an item number which matches the item number (ITEMNO) on an order record.
   e. Why must a resulting indicator be specified in columns 58-59 of the Calculation sheet for the LOKUP operation?

2. a. Design alternating format table input records to create two related tables from the following data. Put one set of entries on each record.

Item Number	Costs
10	$ 10.00
17	75.00
27	125.00
68	1.25
102	.01
700	.05
1640	7.03
2796	72.05
4333	111.11

   b. Define the tables by coding the necessary extension specifications.
   c. Using ITEM as the search word, code the calculation specifications for a two-table search which makes the appropriate cost available.

3. How does a programmer specify that a table search is to satisfy a high, low, or equal condition?

4. What indicates whether a table lookup was successful or not?

5. How does a programmer reference *looked-up* table data in calculation and output specifications?

6. If *looked-up* table data is to be referenced in calculations or output, how can a programmer ensure that the appropriate information will be used?

7. How may table data be changed during execution of an RPG II program?

8. What must be done to change table data permanently (to exist for more than the present run of the program)?

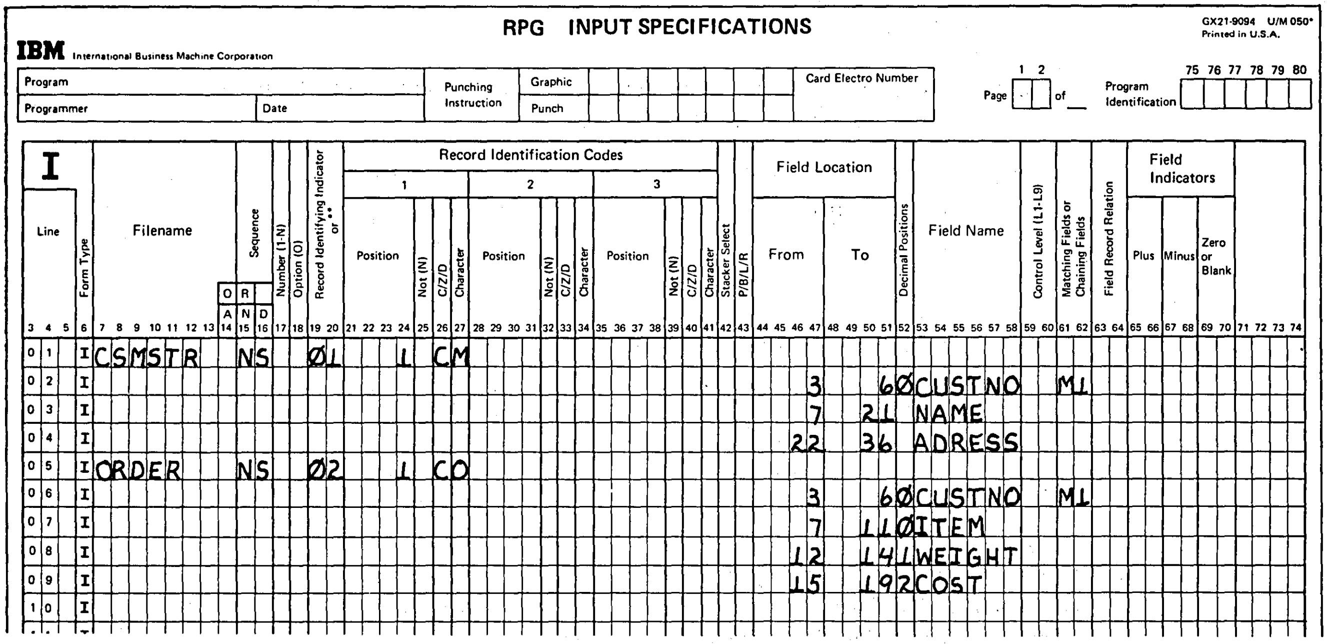
9. What is a short table?

10. What is the advantage of using a short table?

11. Which specification sheets are necessary to program for output of an entire table at end of job? What is the use of each type of specification required?

Review Problem

To perform invoice billing, a corporation processes two input files together in a matching records program. As the following input specifications show, the primary CSMSTR (customer master) file contains a record for every customer who has placed an order, giving the customer number, name, and address. The secondary ORDER file contains data about each order: customer number, item ordered, weight of item, and cost.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P/B/L/R (43)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	(71-74)
01	I	CSMSTR	NS			Ø1	1		C	M																					
02	I																				3	6	Ø	CUSTNO		M1					
03	I																				7	21		NAME							
04	I																				22	36		ADRESS							
05	I	ORDER	NS			Ø2	1		C	O																					
06	I																				3	6	Ø	CUSTNO		M1					
07	I																				7	11	Ø	ITEM							
08	I																				12	14	1	WEIGHT							
09	I																				15	19	2	COST							
10	I																														

According to company policy, merchandise is delivered by truck unless a customer has requested delivery by parcel post. To date, the following 15 customers (by number) have requested parcel post service:

174	2109	596	1157	1475
195	169	456	1366	377
2105	2733	1100	290	1977

The customer always pays parcel post charges. The charge is in accordance with the weight of the ordered item, as follows (assume no order weighs more than 30 pounds):

WEIGHT IN POUNDS	POSTAL CHARGES
1#	$0.45
2	.50
3	.50
4	.55
5	.55
6	.55
7	.60
8	.60
9	.65
10	.65
11	.65
12	.70
13	.70
14	.75
15	.75
16	.75
17	.80
18	.80
19	.85
20	.85
21	.85
22	.90
23	.90
24	.95
25	.95
26	.95
27	1.00
28	1.00
29	1.05
30	1.05

Note: Any fraction of a pound over the weight shown takes the next higher rate.

To produce invoices, the billing program must do the following:

a. Determine if a customer has requested parcel post delivery.
b. If so, determine how much postage is required for the weight ordered.
c. Print the amount of postage due on the invoice (printer output file named INVOICE).

This can best be done by setting up three tables:

- A table of customers who have requested parcel post delivery.

- Two related tables of weights and postal rates.

Your job is to:

1. Design table input records for the three tables. The table of customers requiring parcel post service should be created as a pre-execution time short table to allow for frequent additions and deletions. A table of 24 entries should be sufficient for containing additions. The weight and postal rate tables should be loaded at compile time, since they will not be modified.

2. Define and describe the tables with file description and extension specifications.

3. Code the LOKUP operation(s) on a Calculation sheet to determine how much postage is due, if any.

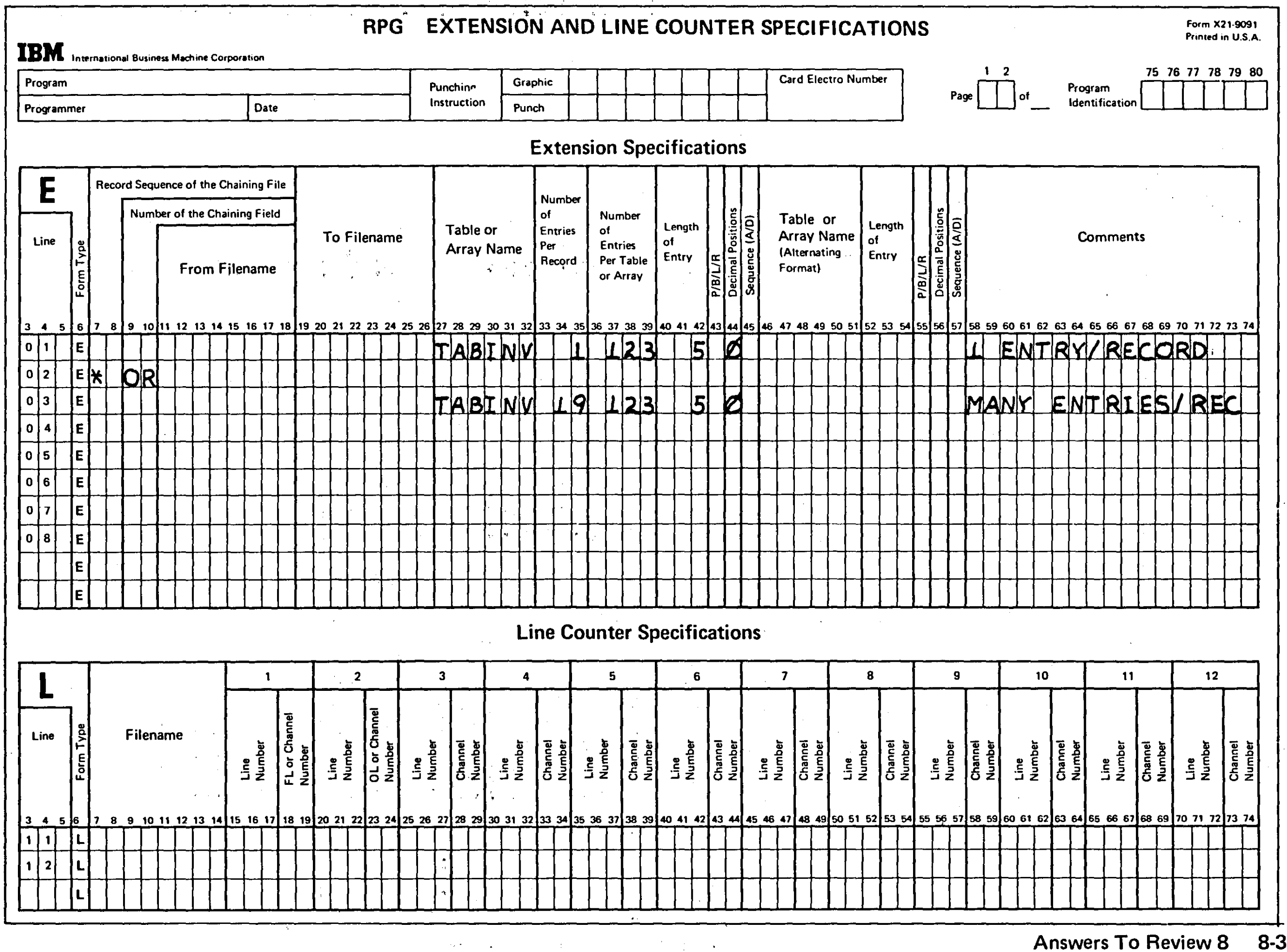
4. Code the output specifications to print the amount of postage due on an order.

1. a. The maximum number of records necessary to contain the table data is 123, with each record containing one item number entry in positions 1-5. The minimum number of records necessary to contain the table data depends on the maximum record length of the device. For 96-column cards, for example, the minimum number is seven records. Records one through six would each contain 19 item number entries punched in columns 1-95. Record seven would contain the remaining nine item number entries punched in columns 1-45. For disk, all entries could reside on a single record, since the maximum record length is 9999.

   b. The table name assigned (for example, TABINV) must meet the following requirements:

   - Three to six characters long
   - May contain any numbers and alphabetic characters (including $, #, @)
   - Must begin with TAB
   - May not contain embedded blanks

   c.



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

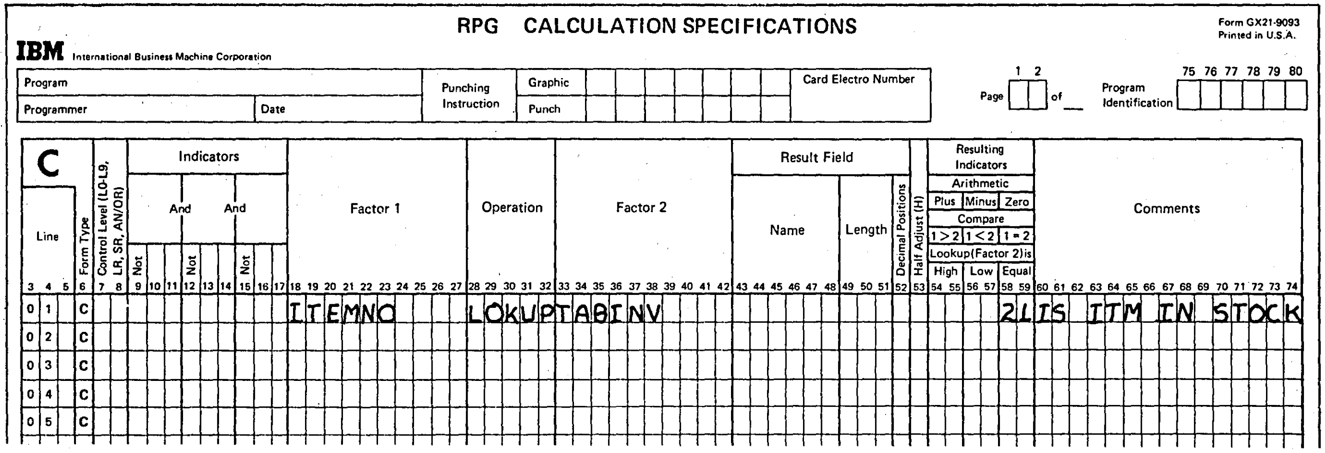
Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					TABINV	1	123	5		Ø							1 ENTRY/RECORD
02	E	*	OR															
03	E					TABINV	19	123	5		Ø							MANY ENTRIES/REC
04	E																	
05	E																	
06	E																	
07	E																	
08	E																	
	E																	
	E																	

Line Counter Specifications

			1		2		3		4		5		6		7		8		9		10		11		12	
Line (3-5)	Form Type (6)	Filename (7-14)	Line Number (15-17)	FL or Channel Number (18-19)	Line Number (20-22)	OL or Channel Number (23-24)	Line Number (25-27)	Channel Number (28-29)	Line Number (30-32)	Channel Number (33-34)	Line Number (35-37)	Channel Number (38-39)	Line Number (40-42)	Channel Number (43-44)	Line Number (45-47)	Channel Number (48-49)	Line Number (50-52)	Channel Number (53-54)	Line Number (55-57)	Channel Number (58-59)	Line Number (60-62)	Channel Number (63-64)	Line Number (65-67)	Channel Number (68-69)	Line Number (70-72)	Channel Number (73-74)
11	L																									
12	L																									
	L																									

d.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Programmer | Date

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not, And Not, And Not (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators Arithmetic Plus / Compare 1>2 / Lookup(Factor 2)is High (54-55)	Minus / 1<2 / Low (56-57)	Zero / 1=2 / Equal (58-59)	Comments (60-74)
01	C			ITEMNO	LOKUP	TABINV							2L	IS ITM IN STOCK
02	C													
03	C													
04	C													
05	C													

e. A resulting indicator must be set on if the item number is in the table to indicate whether the search is successful or not.

2. a. Nine records required for nine pairs of item number/cost entries:

1 001001000

2 001707500

3 002712500

4 006800125

5 010200001

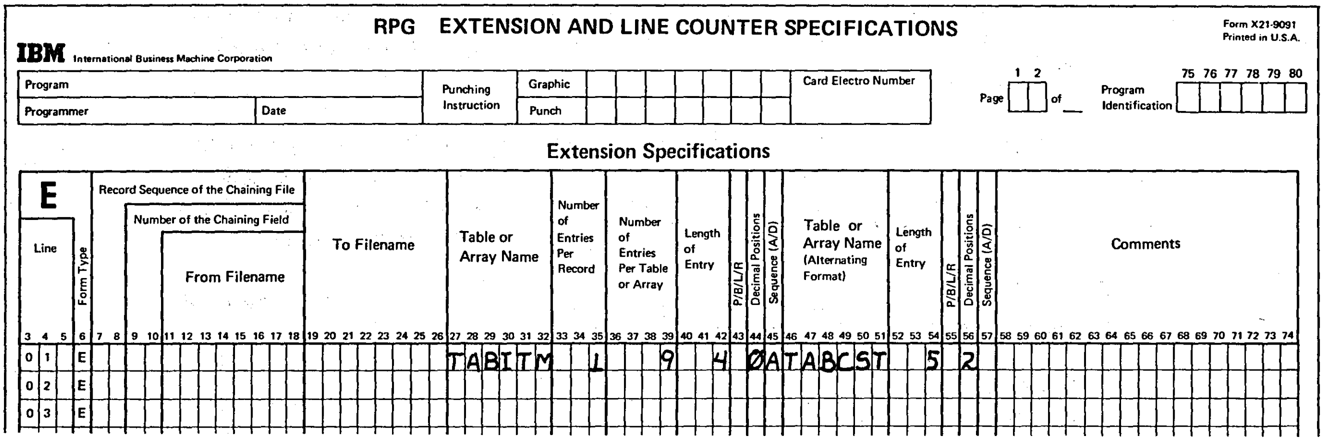
6 070000005

7 164000703

8 279607205

9 433311111

b.



RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

IBM International Business Machine Corporation

Form X21-9091
Printed in U.S.A.

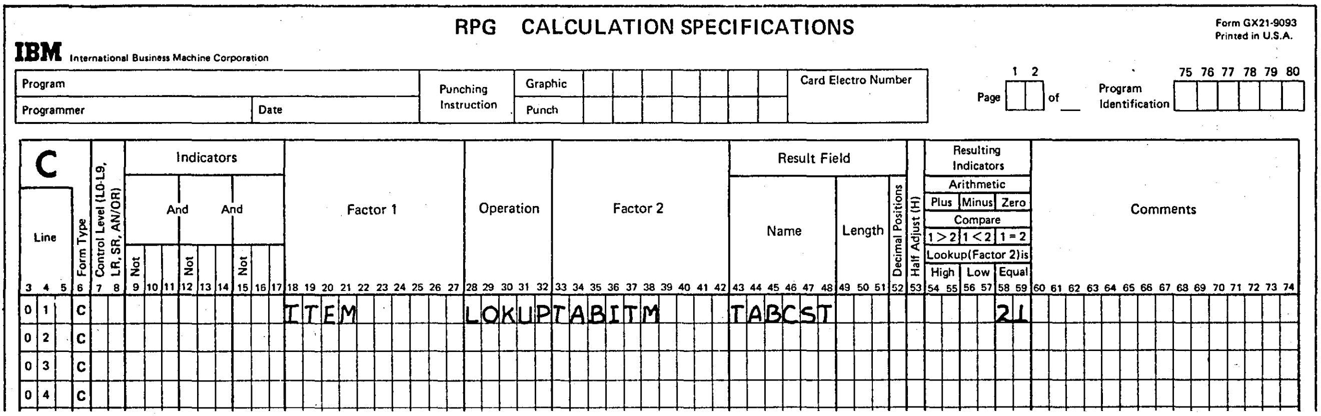
Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Programmer | Date

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					TABITM	1	9	4		Ø	A	TABCST	5		2		
02	E																	
03	E																	

c.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators (Not, And, Not, And, Not)	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic Plus / Compare 1>2 / Lookup(Factor 2)is High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
01	C			ITEM	LOKUP	TABITM	TABCST						21	
02	C													
03	C													
04	C													

3. To search for a high condition enter a resulting indicator in columns 54-55 on the same line of the Calculation sheet as you have specified the LOKUP operation code.

   To search for a low condition, enter a resulting indicator in columns 56-57.

   To search for an equal condition, enter a resulting indicator in columns 58-59.

   If more than one condition will satisfy a search, a resulting indicator should be entered for each condition. The same resulting indicator may be used for both conditions or a different indicator may be specified for each condition.

4. The resulting indicator specified for the high, low or equal condition (columns 54-55, 56-57, or 58-59) will be set on if the particular search condition is satisfied. The indicator will be off if the search was unsuccessful.

5. In any calculation and output specifications which follow a LOKUP operation, use of a table name refers to the table data found by the lookup.

6. The resulting indicator specified with the LOKUP operation can be used to condition calculation and output specifcations that follow the LOKUP.

7. When a table name is specified as the result field of a calculation operation, other than LOKUP, the result of the operation is placed in the table, replacing the last element looked-up. This assumes that a search was previously performed on the specified table.

8. The table input records must be recreated. If calculation specifications change table data during execution, the programmer may code specifications which cause the changed data to be output as new table records. The new output records can then be read in place of the old table input records the next time the program is run.

9. A short table refers to table input records which contain fewer entries than the extension specifications indicate the table can contain.

10. The advantage of using a short table is that entries may be added to a table without changing the extension specifications which describe the table.

11. File Description sheet — assigns the name and device of the output file

Extension sheet — indicates automatic table output at end of job when name of the output file is entered in columns 19-26

Input, Calculation, and Output sheets — not required

Answer to Review Problem

1. *Customer Number Table Input Records*

Since the largest customer number is four digits in length, each entry would require four positions. A customer number less than four digits should be padded in front of the number with zeros or blanks, to form a 4-position entry.

The simplest table input records to design and maintain for this table would contain one entry per record in positions 1-4. With this design, 15 records would be necessary for the 15 customer numbers. To add entries to the table, merely add table input records.

An alternative method is to place the 15 entries in positions 1-60 of a single record. A record can contain a maximum of 24 4-digit entries. Entries may be added to the table by placing new customer numbers in unused positions of the same record. However, to delete entries, the entire record would have to be recreated.

Weight and Postal Rate Table Input Records

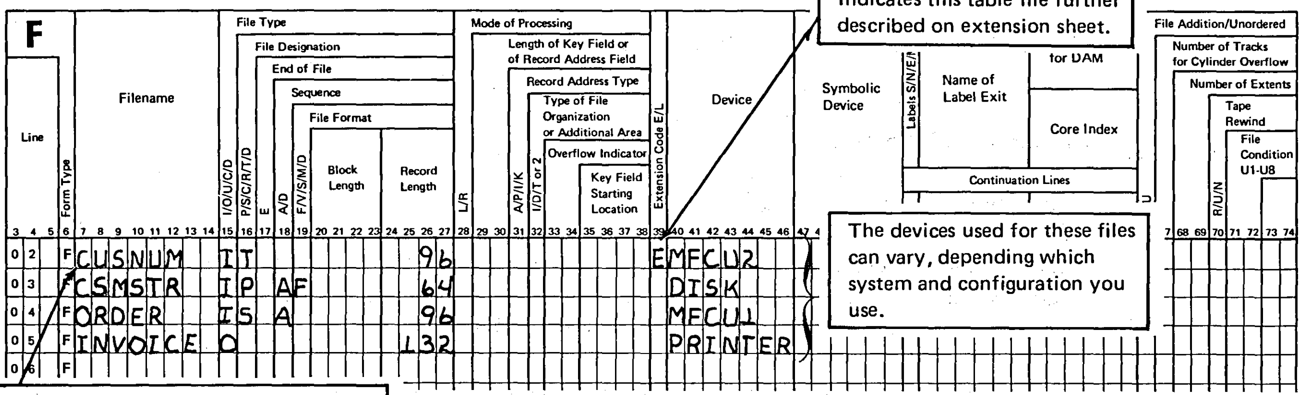
The WEIGHT field from the ORDER input file is to be used as the search word for locating the appropriate weight in the table. Therefore, the weight table entries must be the same length and format as the search word: three digits in length with one decimal position. The corresponding postal rates can be three digits with two decimal positions, to accommodate the largest entry 1.05.

Using an alternating format, one record could be used for each pair of entries (positions 1-6), for a total of 30 table input records. Otherwise, the first 16 pairs of entries could be contained in positions 1-96 of one record with the remaining 14 pairs contained in positions 1-84 of a second record.

If separate records are to be used for each table, the 30 weight entries could be in columns 1-90 of one record, and the 30 postal rate entries in columns 1-90 of another record. Using separate table format, 30 records would be required for each table, if each record contained only one entry.

2. Specifications to define and describe customer number table:

File Description Specification



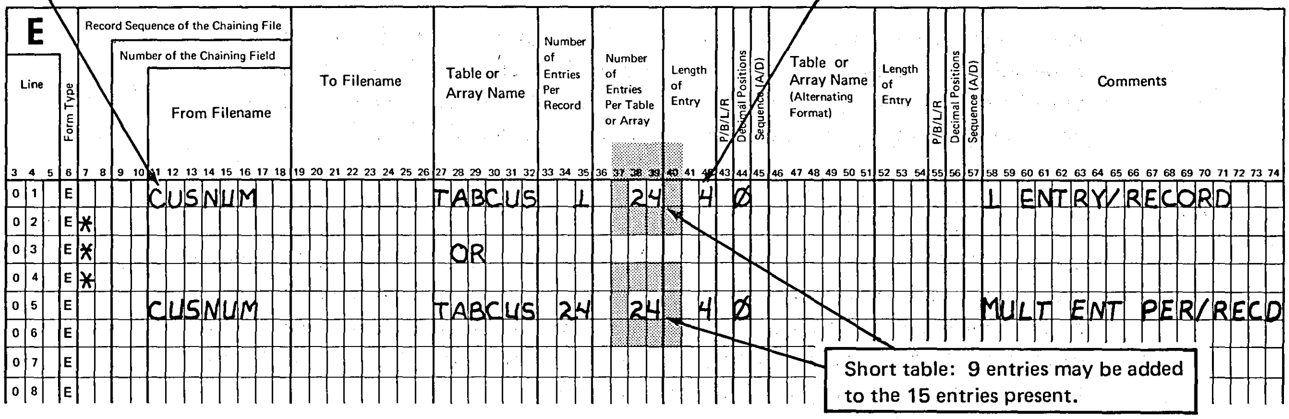
Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V/S/M/D	Block Length	Record Length	Extension Code E/L	Device
02	F	CUSNUM	I	T					96	E	MFCU2
03	F	CSMSTR	I	P		A	F		64		DISK
04	F	ORDER	I	S		A			96		MFCU1
05	F	INVOICE	O						132		PRINTER
06	F										

Indicates this table file further described on extension sheet.

The devices used for these files can vary, depending which system and configuration you use.

Pre-execution time table file containing table-input records must be loaded before other data files.

Extension Specifications



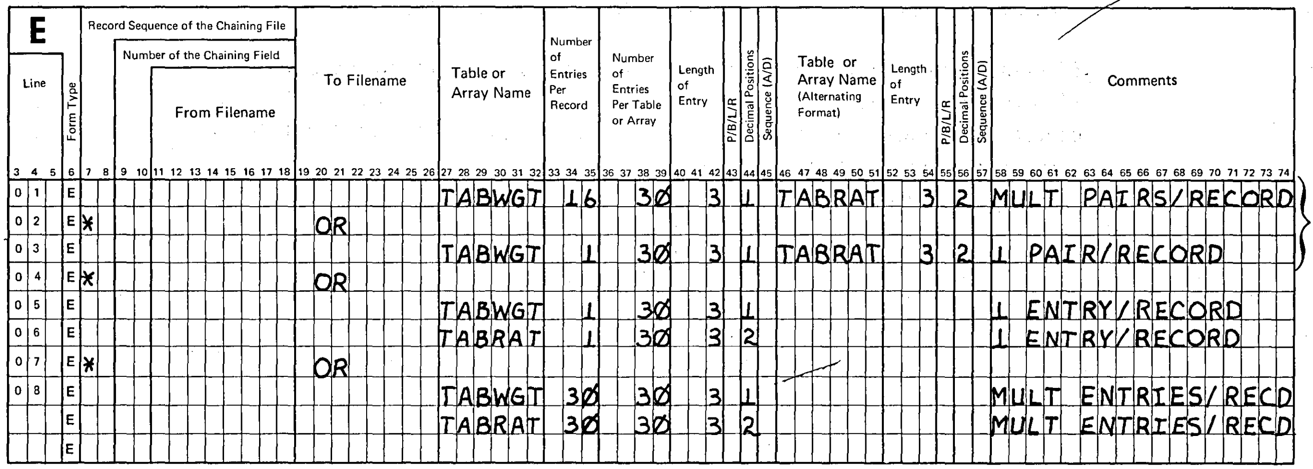
Line	Form Type	7	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E		CUSNUM		TABCUS	1	24	4		Ø							1 ENTRY/RECORD
02	E	*															
03	E	*			OR												
04	E	*															
05	E		CUSNUM		TABCUS	24	24	4		Ø							MULT ENT PER/RECD
06	E																
07	E																
08	E																

Same length as search word.

Short table: 9 entries may be added to the 15 entries present.

Specifications to define weight and postal rate tables:

Extension Specifications



Line	Form Type	7	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E				TABWGT	16	3Ø	3		1		TABRAT	3		2		MULT PAIRS/RECORD
02	E	*		OR													
03	E				TABWGT	1	3Ø	3		1		TABRAT	3		2		1 PAIR/RECORD
04	E	*		OR													
05	E				TABWGT	1	3Ø	3		1							1 ENTRY/RECORD
06	E				TABRAT	1	3Ø	3		2							1 ENTRY/RECORD
07	E	*		OR													
08	E				TABWGT	3Ø	3Ø	3		1							MULT ENTRIES/RECD
	E				TABRAT	3Ø	3Ø	3		2							MULT ENTRIES/RECD
	E																

Alternating Format

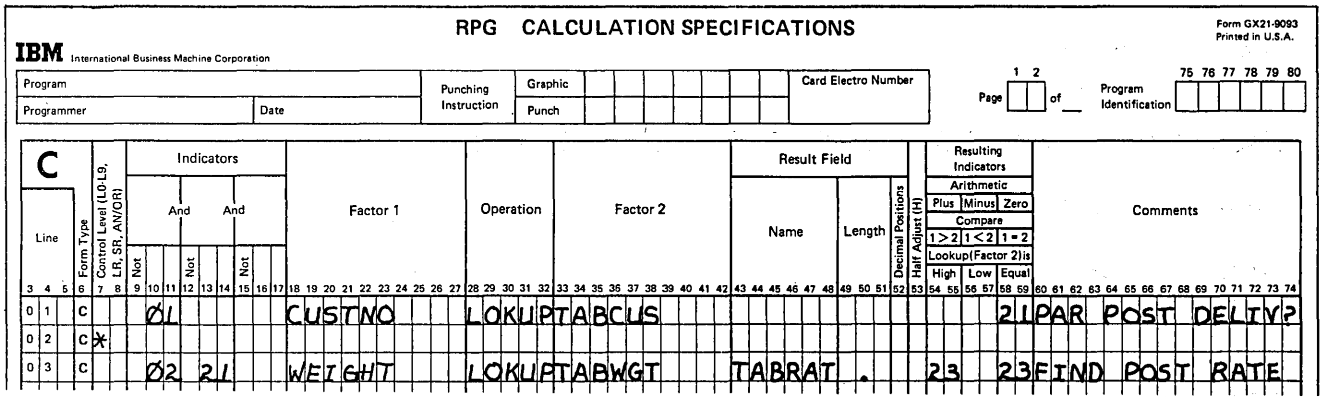
No entry indicates compile time table

File Description sheet not required since they are compile time tables.

3. Calculation specifications for LOKUP operations:

Line 01 – When a CSMSTR record is read (01 on), use CUSTNO field as search word to determine if that customer requires parcel post service. Indicator 21 set on if customer number in table.

Line 03 – When an ORDER record is read (02 on) and that customer requires parcel post (21 on from successful search), then search weight table using WEIGHT field as search word. Indicator 23 set on when correct weight found (same number of pounds or next higher entry if weight is in fractions of pounds). When 23 is set on, correct postal charge is available from postal rate table.



IBM International Business Machine Corporation

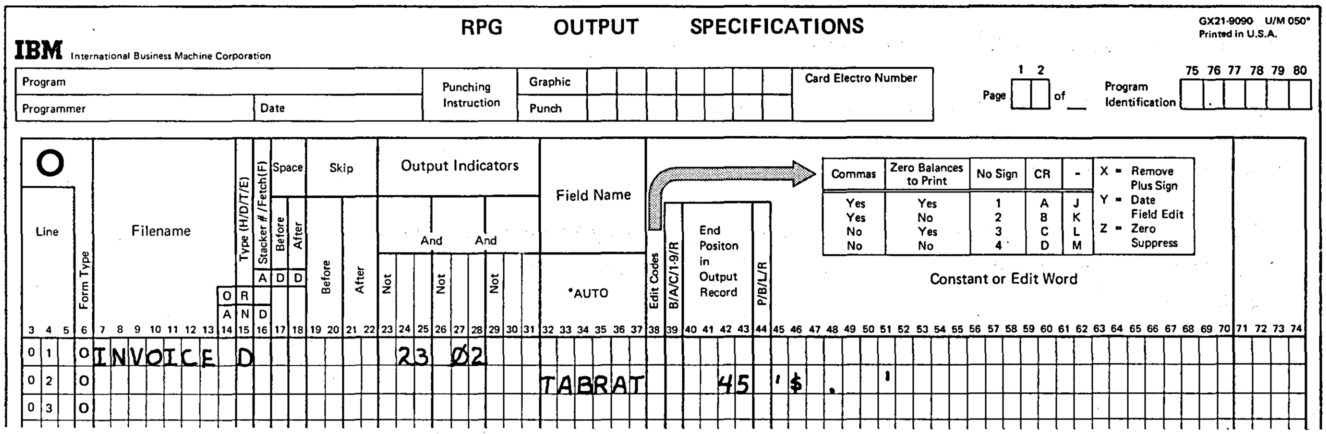
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

```
01 C   Ø1      CUSTNO    LOKUPTABCUS                   21PAR POST DELIV?
02 C*
03 C   Ø2 21   WEIGHT    LOKUPTABWGT    TABRAT .   23  23FIND POST RATE
```

4. Output specifications to print postal charge on invoice:

Line 01 – If customer required parcel post and correct postage charge has been determined (23 on), print the postage chart from the postal rate table, TABRAT.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

```
01 OINVOICE D        23 Ø2
02 O                         TABRAT     45 '$ .  '
03 O
```

Chapter 9. Arrays

CHAPTER 9 DESCRIBES:

Use of arrays and RPG II coding to reference an entire array or individual elements of the arrays.

XFOOT operation code.

LOKUP operation code.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Use of and coding for tables.

Exception output.

RPG II object cycle.

OR relationship.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Determine the use of arrays as opposed to the use of tables.

Define an array on the Extension sheet.

Code problems that reference all elements in an array.

Code problems referencing individual elements in an array.

Define and code the LOKUP operation code with arrays.

Describe data and store it in an array.

Note: You can use the review questions contained in *Review 9* at the end of this chapter to test your comprehension of the chapter. Answers follow the review questions.

INTRODUCTION

An array is a continuous series of data fields stored side by side so they can be referenced as a group. In an array, each individual data field is called an *element*. Figure 9-1 shows an array of 12 elements containing the total sales for each month of the year. Each element of the array has the same characteristics; that is, each contains data in the same format (alphameric or numeric), of the same length, and with the same number of decimal positions. An array element may be positive, negative, or unsigned; within a numeric array, elements may be positive or negative.

An array is very similar in concept to a table. Both arrays and tables are set up by coding extension specifications. The type of data which you can put in an array is the same as that which you can put in a table. The data can be punched on cards, keyed in by the operator, or written on disk or tape. The data can be loaded into an array at compilation time or just before execution time. An array can also be built from data extracted from normal input files or from data produced during the program as a result of calculations. The way data is arranged in storage is the same for tables and arrays; one element of data immediately follows another. The uses, however, of tables and arrays differ considerably.

WHEN TO USE AN ARRAY INSTEAD OF A TABLE

In most cases, tables contain constant data such as tax rates, shipping instructions, or discount rates. The constant data is then used for calculations or printing with variable transaction data. Arrays are generally used for variable data and totals which are used independently of the variable transaction data.

You should use arrays instead of tables when you want to reference all elements at one time. Arrays can reduce the number of RPG II specifications you must code for such a program, as well as the time required to reference the entries. Arrays should also be used when you are able to directly reference a data item within a group of items and do not need to do a look-up based on a search word.

DEFINING AN ARRAY

You tell the RPG II program that you wish to set up an array by coding extension specifications in much the same way as you would code them for tables. As shown in Figure 9-2, coding on the Extension sheet varies slightly, depending on when the array data is to be read into the array that is set up by the RPG II compiler. Array data can be stored in the array at three different times (see also *Loading Arrays*):

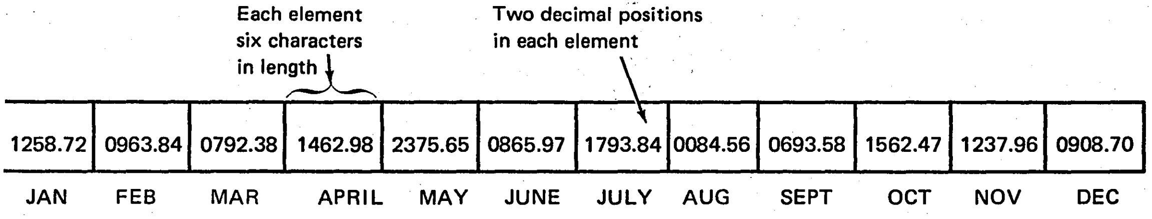
1. *Compile time.* The array records immediately follow, and are compiled with, the source program. (If you have a Card System, both array data and source program are loaded from the secondary hopper.)

2. *Pre-execution time.* The array records are read like any other data file, except that they are all read before any processing is done. (If you have a Card System, both array records and object program are loaded from the secondary hopper.)

3. *Execution time.* The array is loaded from information in input records or data generated by calculations.

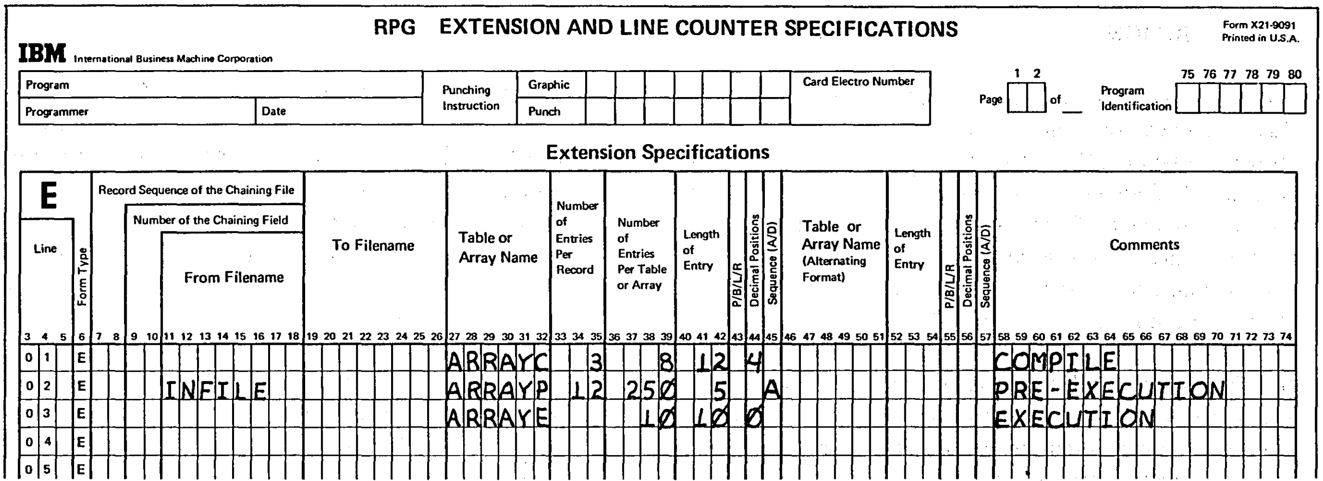
The following Extension sheet entries are used to define and describe arrays (see Figure 9-2):

Columns 11-18 (From Filename): Pre-execution time arrays are read from an input file similar to other data files. The name of the file containing the array must be entered in columns 11-18 of the Extension sheet. This file must be designated as a table file in column 16 of the File Description sheet. A table file is read completely and data is loaded into the array before execution of the program begins. The same MFCU file can be named in these columns for more than one array. Input files on other devices can contain only one array. On the Card System, pre-execution time arrays must be loaded from the secondary MFCU hopper.



1258.72	0963.84	0792.38	1462.98	2375.65	0865.97	1793.84	0084.56	0693.58	1562.47	1237.96	0908.70
JAN	FEB	MAR	APRIL	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC

Figure 9-1. 12-Element Numeric Array



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					ARRAYC	3	8	12		4							COMPILE
02	E			INFILE		ARRAYP	12	25Ø	5			A						PRE-EXECUTION
03	E					ARRAYE		1Ø	1Ø		Ø							EXECUTION
04	E																	
05	E																	

Figure 9-2. Defining Arrays

Columns 19-26 (To Filename): If you want your entire array to be written to an output file at end of job, enter the name of the output file in columns 19-26. You cannot use the To Filename entry to write execution time arrays to an output file; instead you must use output specifications (see *Output of an Entire Array*, later in this chapter).

Columns 27-32 (Table or Array Name): All arrays used in your program must be assigned a name of six characters or less which is entered in columns 27-32. The rules for naming arrays are similar to those for naming tables; an array name can consist of any combination of alphabetic characters and numbers. However, while the first character must be an alphabetic character, an array name cannot begin with the letters TAB. This is the way the compiler distinguishes between an array and a table.

Columns 33-35 (Number of Entries Per Record): For compile time and pre-execution time arrays, enter the number of array elements in each input record. These columns must be blank for execution time arrays.

Columns 36-39 (Number of Entries Per Table or Array): These columns are used to enter the number of elements in the array (from 1 to 9999). This number should be entered so that the last digit is in column 39.

Columns 40-42 (Length of Entry): The length of each element (number of characters, including blanks) should be specified in columns 40-42, with the number ending in column 42. The length, which must be the same for every element in the array, cannot be greater than 255.

Column 43 (Packed/Binary): Disk or tape users may specify in column 43 that pre-execution time array data is in packed decimal or binary format. On a disk system, 80-column card users can specify packed pre-execution time data.

Column 44 (Decimal Positions): If the elements in an array are numeric, the number (0-9) of digits to the right of the decimal point should be entered in column 44. Even if no decimal positions are present, a zero must be specified if the elements are to be considered numeric. A blank in column 44 indicates that the elements are to contain alphameric data. Remember, however, that if arithmetic operations are to be performed on the elements, the array must be defined as numeric.

Column 45 (Sequence): If your array data is in sequence, enter A (ascending) or D (descending) in column 45. Sequence is not checked for execution time arrays, but this column must contain an entry if high or low look-up is used (see *LOKUP of an Array*).

Columns 46-57 (Alternating Arrays): Columns 46 through 57 are used if two related arrays are set up in an alternating format on input records. Alternating arrays cannot be described with execution time arrays.

The extension specifications only reserve the appropriate space in storage for the array. In a following section, you will learn how data is stored in the array.

REFERENCING ALL ELEMENTS IN AN ARRAY

Suppose a company employs 15 sales clerks whose daily sales are recorded on a punched card (SALES). As Figure 9-3 shows, field 1 contains sales for clerk #1, field 2 for clerk #2, and so on. There is one SALES record for each day. In addition to a daily amount, the company wishes to have a monthly sales total for each clerk. Therefore, at the end of the month, the daily sales amounts for a clerk must be accumulated.

As shown in Figure 9-4, an array (MONTH) of 15 elements is set up to contain the monthly totals. Another array, called DAY, could be set up to contain the 15 sales amounts for one particular day. The daily sales record is read and each clerk's *sales amount* is placed in the appropriate element of array DAY.

Array to Array Calculations

Once the first SALES record is read and the data stored in the DAY array, the 15 elements of DAY are added to the 15 elements of MONTH. In other words, element 1 of DAY is added to element 1 of MONTH, element 2 to element 2, and so on (Figure 9-5).

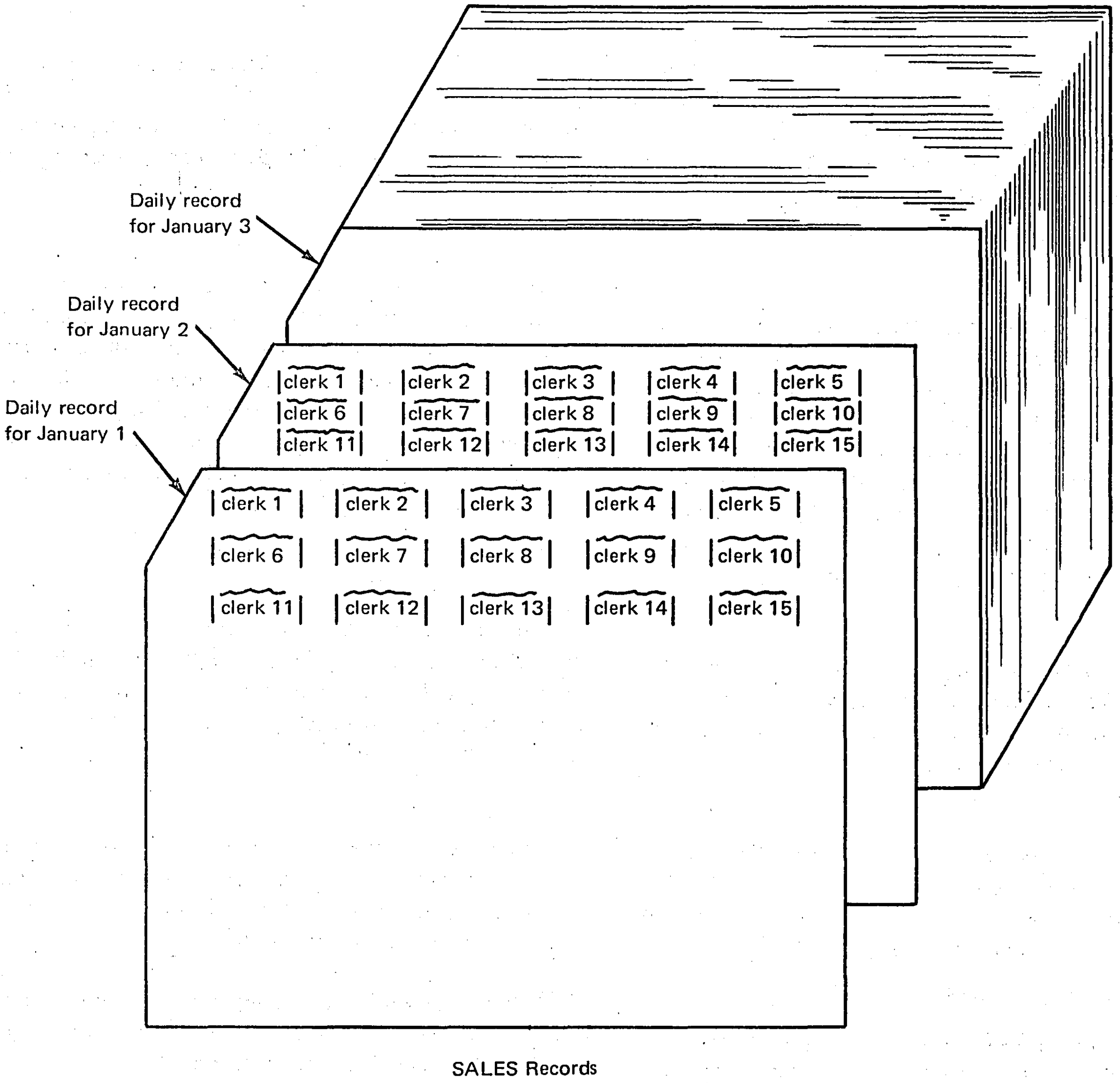


Figure 9-3. SALES Records

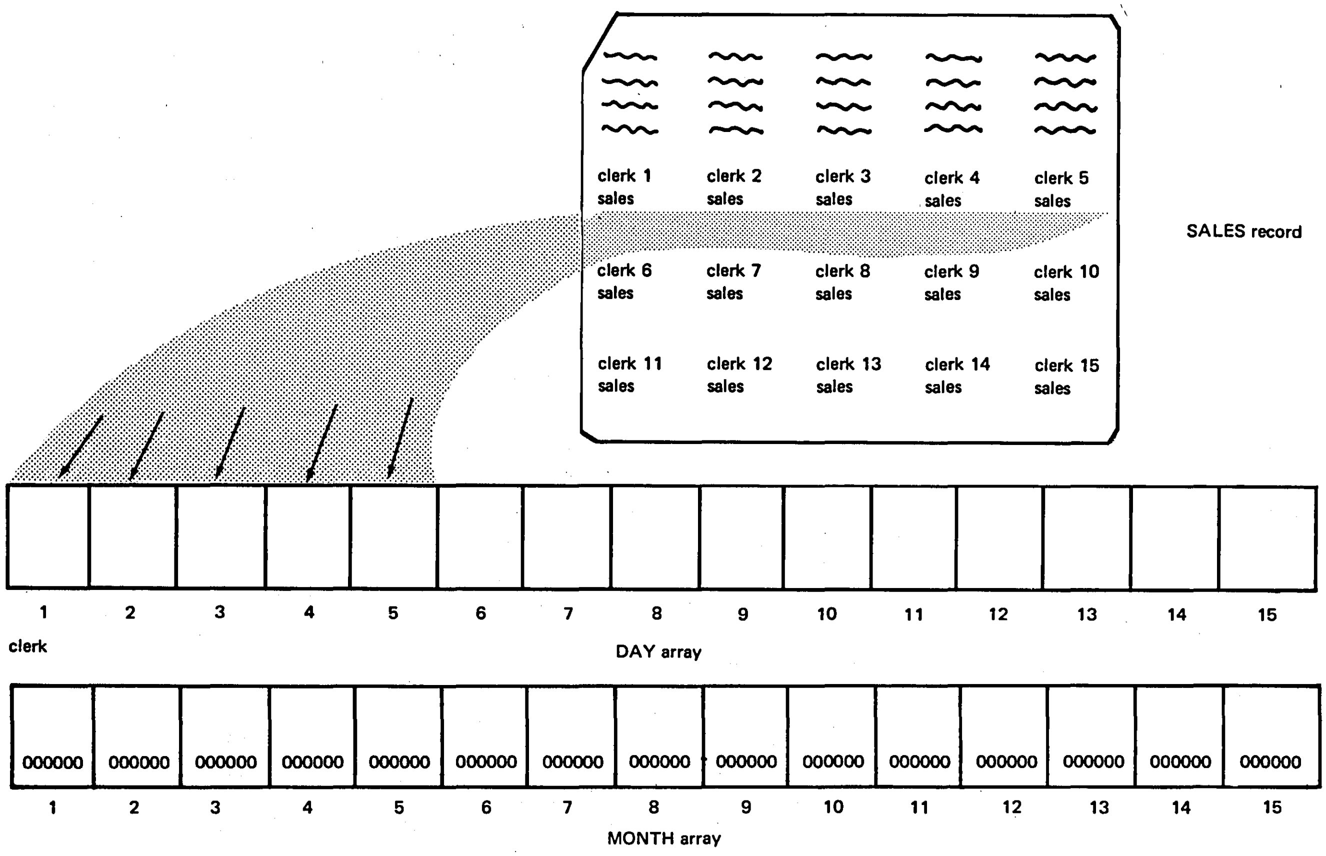
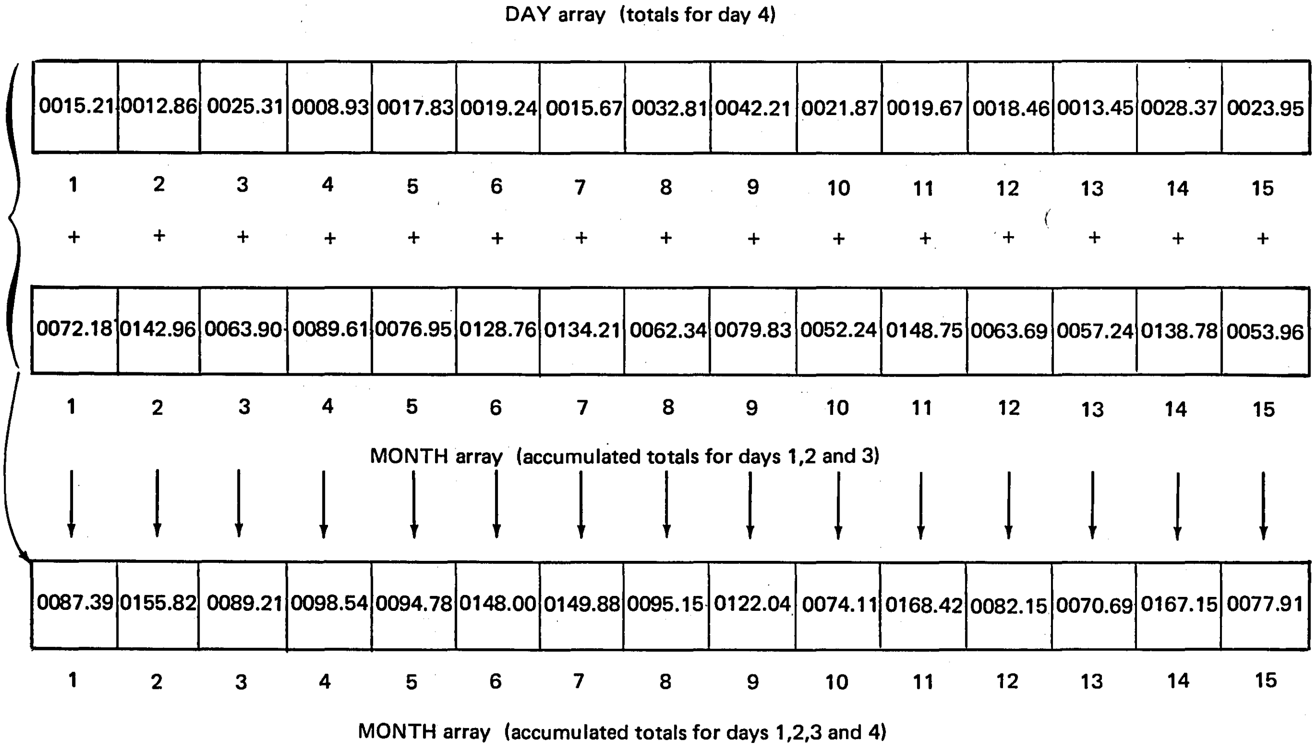


Figure 9-4. Using Arrays to Contain Sales Data



DAY array (totals for day 4)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0015.21	0012.86	0025.31	0008.93	0017.83	0019.24	0015.67	0032.81	0042.21	0021.87	0019.67	0018.46	0013.45	0028.37	0023.95

+ \+ \+ \+ \+ \+ \+ \+ \+ \+ \+ \+ \+ \+ \+

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0072.18	0142.96	0063.90	0089.61	0076.95	0128.76	0134.21	0062.34	0079.83	0052.24	0148.75	0063.69	0057.24	0138.78	0053.96

MONTH array (accumulated totals for days 1,2 and 3)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0087.39	0155.82	0089.21	0098.54	0094.78	0148.00	0149.88	0095.15	0122.04	0074.11	0168.42	0082.15	0070.69	0167.15	0077.91

MONTH array (accumulated totals for days 1,2,3 and 4)

Figure 9-5. Adding One Array to Another Array

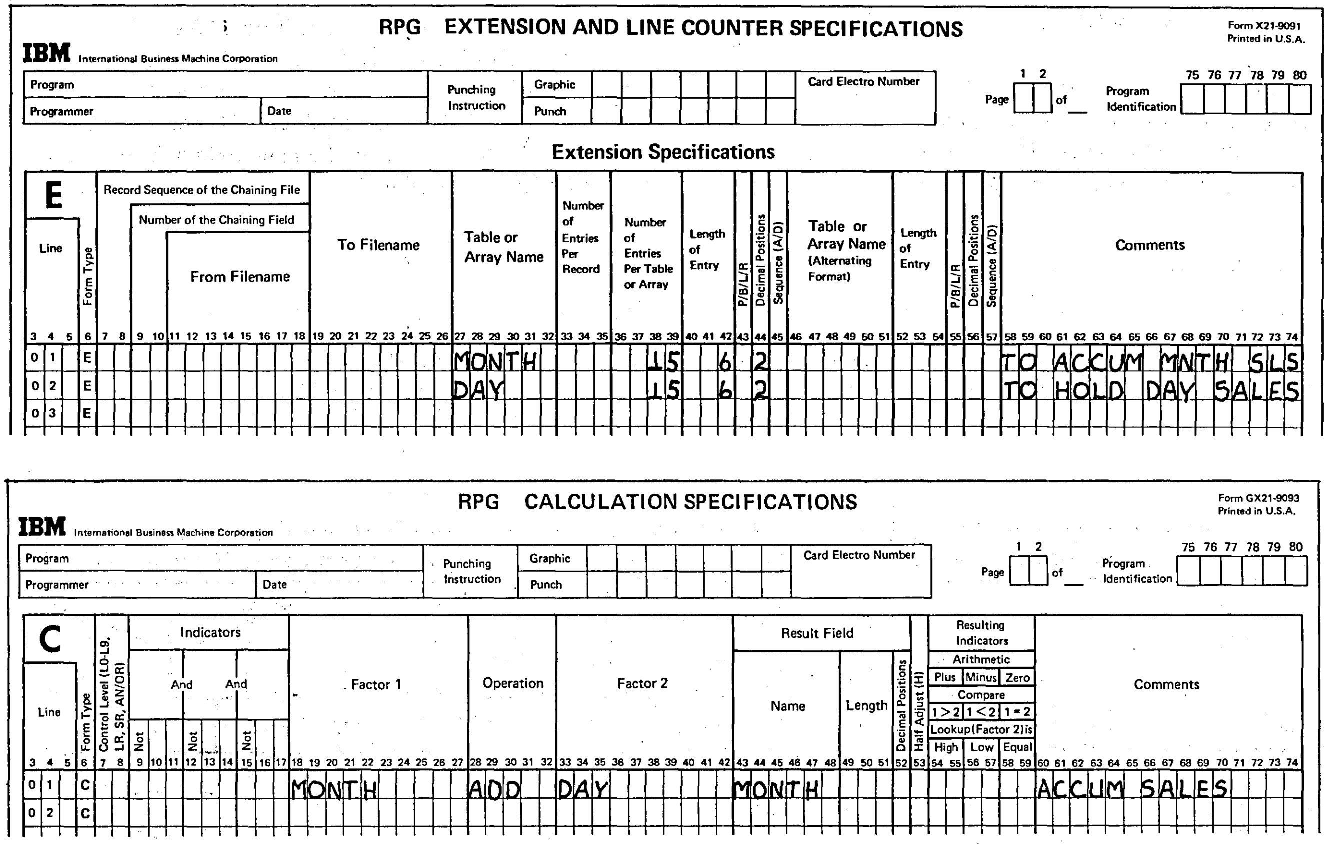
The 15 accumulated sale amounts (results of the additions) are stored in MONTH. Then, another SALES card is read into the DAY array. The new DAY fields are then added again to the accumulated totals in MONTH.

This method is similar to using two tables and adding an entry from one table to an entry in the other table. However, performing the operations using tables requires more specifications than to doing the job using arrays.

With tables, you must reference each element (sales amount for a clerk) separately. First, you must perform a table lookup to find the appropriate sale amount from the day table. Of course, since you do not know the amount of each sale, you cannot search the day table directly. A related table of sales clerk numbers must be set up and searched. Only after you find the appropriate sales clerk entry is the corresponding sale amount in the day table made available. Then you must lookup the corresponding element of the month table. At this point, use of the table names in calculations or output would finally refer to each of the entries looked up. An addition operation would then be required to add the two entries and place the result in the month table. After all this, you have accumulated a total for only one of the sales clerks.

To repeat the same procedure 14 more times for the other sales clerks' entries, the program must read 14 records and go through 14 program cycles. This occurs when you use a table name in specifications. The name refers to only one element, the entry just looked up.

On the other hand, if you have defined your groups of data as arrays rather than tables, only one calculation specification is necessary. The name of an array actually refers to all of the elements in that array. Adding the array DAY to the array MONTH causes every element of one array to be added to corresponding elements of the other array (1 to 1, 2 to 2, 3 to 3, etc.). Since the MONTH array is specified under Result Field, the result of each addition is placed back into the appropriate element of MONTH (Figure 9-6).



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					MONTH		15	6		2							TO ACCUM MNTH SLS
02	E					DAY		15	6		2							TO HOLD DAY SALES
03	E																	

IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not (9)	(10-11)	And: Not (12)	(13-14)	And: Not (15)	(16-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field: Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus, Compare 1>2, Lookup (Factor 2) is High (54-55)	Minus, 1<2, Low (56-57)	Zero, 1=2, Equal (58-59)	Comments (60-74)
01	C								MONTH	ADD	DAY	MONTH							ACCUM SALES
02	C																		

Figure 9-6. Referencing All Elements of an Array

Notice on the Calculation sheet in Figure 9-6 that no resulting indicators have been specified for this arithmetic operation. When an array name is specified in a calculation, the operation is performed on every element of the array. Therefore, there are a multiple number of results; in this case, 15 sales totals. A resulting indicator can indicate the condition of only a single result. Thus, resulting indicators cannot be used when referencing an entire array as a Result Field. There are two exceptions when resulting indicators can be used, as explained under *Adding All Elements Within An Array and Searching An Array For A Particular Element.*

Operations Which Can be Performed on Arrays

As mentioned, an operation to be performed on an array is performed for every element in the array. A result is then produced for each element operated on. For this reason, certain operations cannot be performed on arrays, because the results have no meaning. The operation codes COMP (compare), TESTZ (test zone), MVR (move remainder), TESTB (test bit), BITON (set bits on), BITOF (set bits off), and DSPLY (display) cannot be used with an array.

Performing Operations on Arrays of Different Lengths

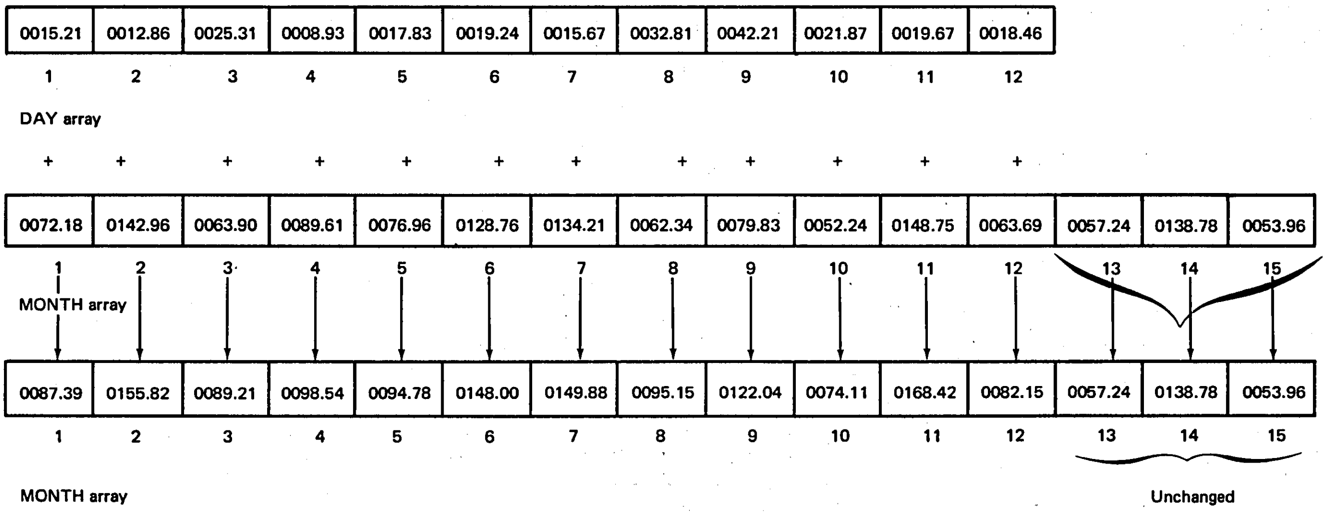
In the last example, all arrays used in an operation were of the same length; Factor 1, Factor 2, and the result array each contained 15 elements. Thus the operations were carried out until all elements were processed.

Suppose, as shown in Figure 9-7, that DAY only contains 12 elements while the MONTH array contains 15 elements. In such a case, the operations are performed only until the last element in the shortest array has been processed. Thus, the 12 elements of DAY are added to the first 12 elements of MONTH, and the 12 results are placed in the first 12 elements of MONTH. The remaining three elements of the result field (MONTH) remain unchanged. Likewise, if the result array is shorter than any of the factors (arrays), the operation is repeated only for the number of elements in the shortest (result) array.

Calculations Using Arrays and Single Fields (or Constants)

Another way in which you can perform calculations on an entire array is by adding (or multiplying, etc.) the same value to every element in the array. For example, suppose the sales clerks are to receive a commission of 10 percent of their sales, to be paid at the end of the month. After all daily sales have been accumulated into the MONTH array, you want to multiply each of the 15 elements in MONTH by the value .10 and to place the commission amounts in another 15-element array called COMMIS.

To do this, it is not necessary to set up a 15-element array for the commission rates, with each element containing the value .10. In an array operation, when one of the factors is a field (containing a value) or a constant, the operation is performed using the same field or constant on every element in the array.



0015.21	0012.86	0025.31	0008.93	0017.83	0019.24	0015.67	0032.81	0042.21	0021.87	0019.67	0018.46
1	2	3	4	5	6	7	8	9	10	11	12

DAY array

0072.18	0142.96	0063.90	0089.61	0076.96	0128.76	0134.21	0062.34	0079.83	0052.24	0148.75	0063.69	0057.24	0138.78	0053.96
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

MONTH array

0087.39	0155.82	0089.21	0098.54	0094.78	0148.00	0149.88	0095.15	0122.04	0074.11	0168.42	0082.15	0057.24	0138.78	0053.96
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

MONTH array

Figure 9-7. Operations on Arrays of Different Lengths

You can also use a field or constant as both factors to place the same result in every element of an array. The calculation specifications in Figure 9-8 show the single field named DISCNT being subtracted from the single field AMOUNT, with the result placed in a 5-element array named DUE. The value (017) in DISCNT is subtracted from the value (243) in AMOUNT, and the result (226) is placed in each of the five elements of the DUE array.

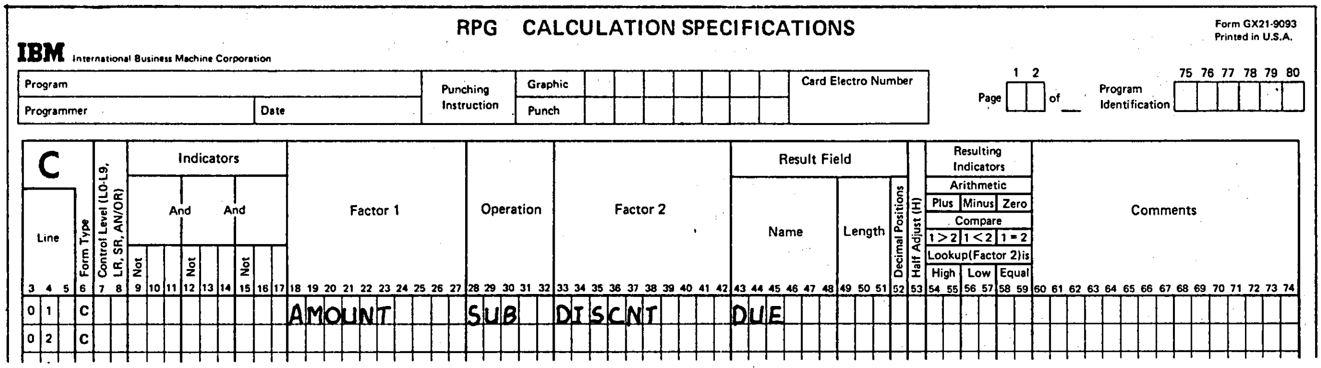
Adding All Elements Within An Array

In accumulating a monthly sales total for each clerk, the amount each clerk sold was determined. Suppose, in addition, the company also wants to know the total of all sales each day.

As mentioned before, each clerk's daily sales are stored in a separate element of a 15-element array named DAY. To obtain a total of all sales for the day, you must add together the contents of all elements in the array. The sum can then be placed in a single field.

The XFOOT operation code (Figure 9-9, columns 28-32) tells the computer to sum the contents of every element in the array named in Factor 2. Columns 18 through 27 (Factor 1) of the Calculation sheet are left blank since the XFOOT operation involves only the values in one array. The sum of the DAY array elements is then placed in the single field named in columns 43 through 48 (Result Field).

In most types of array calculations, multiple results are produced in accordance with the number of elements in an array. However, performing an XFOOT operation provides only one result, the total of all elements. For this reason, you specify a single field name rather than an array name, under Result Field. Furthermore, since there is only one result, a resulting indicator may be assigned in columns 54-59 to determine if the sum is plus, minus, or zero. In this case, a resulting indicator was not specified (Figure 9-9) because the sales amounts will always be positive.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program	Punching Instruction	Graphic	Card Electro Number	Page 1 2 of
Programmer	Date	Punch		Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / Minus / Zero; 1>2 / 1<2 / 1=2; High / Low / Equal (54-59)	Comments (60-74)
01	C			AMOUNT	SUB	DISCNT	DUE					
02	C											

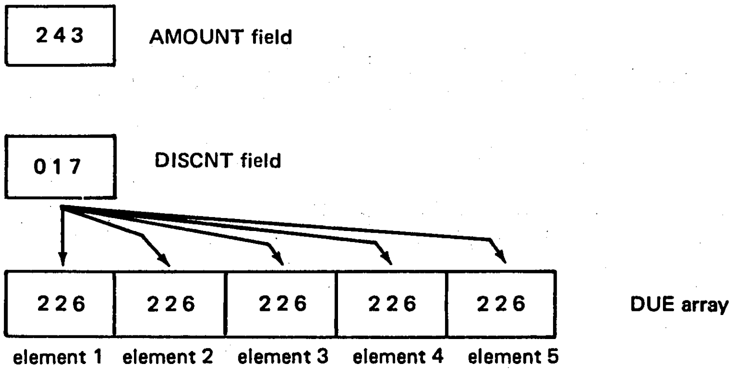
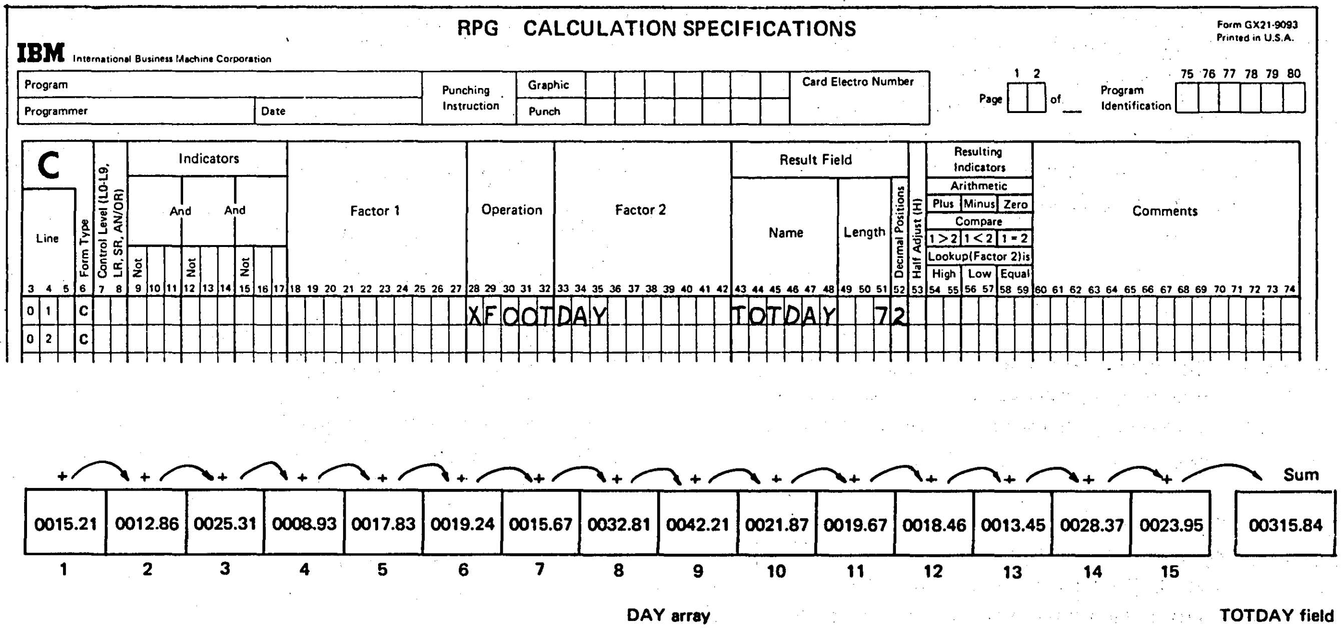


Figure 9-8. Storing the Same Data in All Array Elements



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page □□ of ___ | Program Identification □□□□□□

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C				XFOOT	DAY	TOTDAY	7	2			
02	C											

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	TOTDAY field
0015.21	0012.86	0025.31	0008.93	0017.83	0019.24	0015.67	0032.81	0042.21	0021.87	0019.67	0018.46	0013.45	0028.37	0023.95	00315.84

Figure 9-9. Adding All Elements of an Array

Output of an Entire Array

You may want to have an entire array written or punched out. Perhaps you want to look at the contents of the array at some point during the program run or at the end of the run. Or, you may want to have array elements put out to be used as input the next time the program is run. Output of an entire array can be specified in two ways, with extension specifications or with output specifications.

By Extension Specifications

Like tables, compile time arrays and pre-execution time arrays can be written out at end of job by simply entering the name of the output file under To Filename (columns 19-26) on the Extension sheet (see Figure 9-10). The output file must be named on the File Description sheet, but no output specifications are necessary. The entire array, including unused elements (blanks or zeros), is put out.

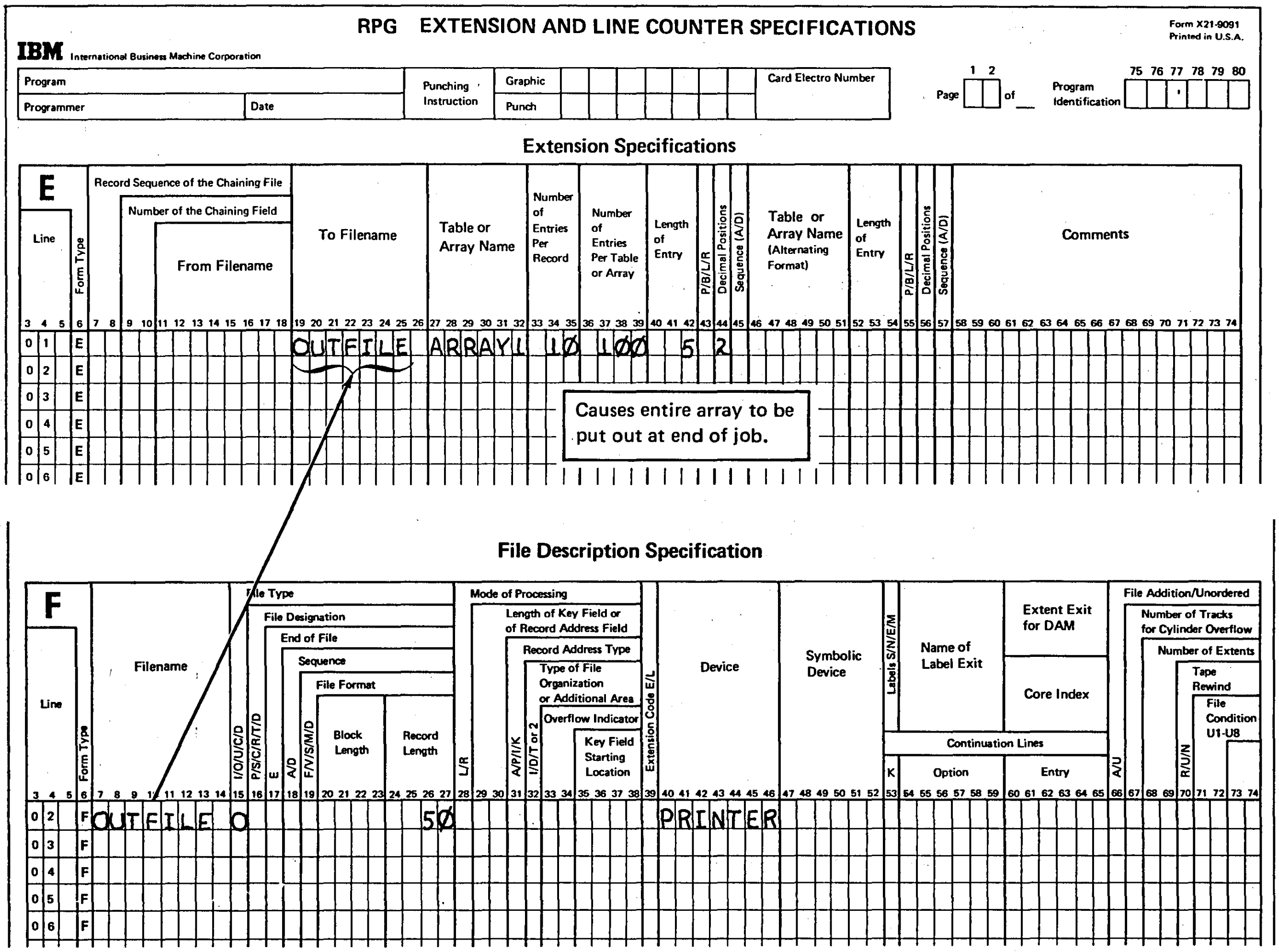
With the Extension specifications shown in Figure 9-10, the array, ARRAY1, is put out automatically at end of job.

The data output may not be exactly the same as the data input. Your program can change or update array entries, causing modified data to be put out, not the original data. Except for printer output, however, the format of the array output records will be the same as the input records.

By Output Specifications

The second method of specifying output of an entire array is with output specifications. By specifying the array name under Field Name (columns 32-37) on the Output sheet, all elements within the named array are punched, printed, or written on the indicated output file (Figure 9-11). All types of arrays, compile, pre-execution, and execution, can be put out using output specifications.

Any output conditioning indicators specified in columns 23 through 31 of the Output sheet determine when during the program the array elements will be printed or punched. If no indicators are specified, the entire array is printed or punched every time a record is processed. Indicators can be specified to put out array data during detail cycles or at total time. You may want to put out array data at total time by customer or inventory item, for example, to be used as input to subsequent update runs.



RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Extension Specifications

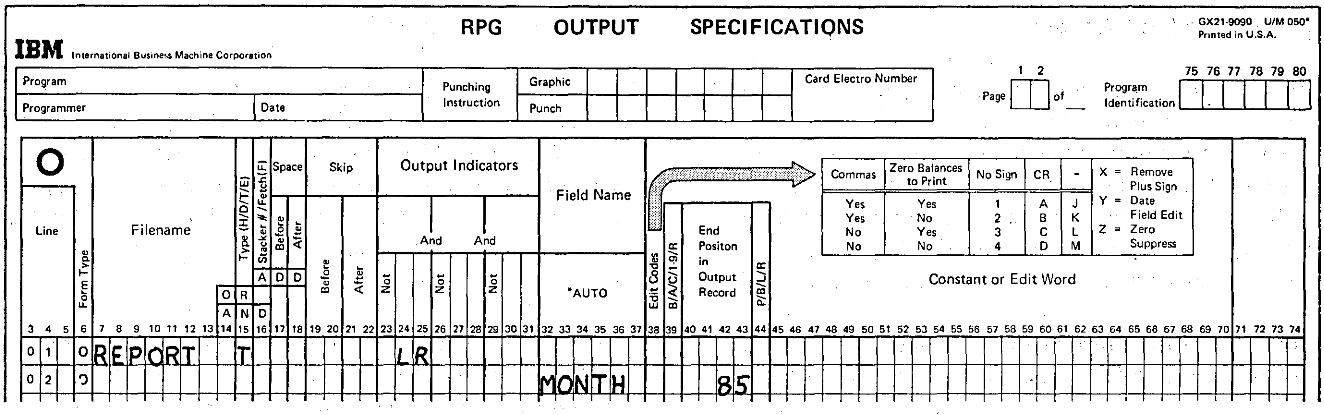
Line	Form Type	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	Decimal Positions
01	E	OUTFILE	ARRAY1	10	100	5	2

Causes entire array to be put out at end of job.

File Description Specification

Line	Form Type	Filename	File Type	Record Length	Device
02	F	OUTFILE	O	50	PRINTER

Figure 9-10. Specifications for Output of an Entire Array at End of Job



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record
01	O	REPORT	T	LR		
02	O				MONTH	85

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Figure 9-11. Output of an Entire Array

RPG II determines where the array data is to be put out on a card, printer file, disk or tape by the end position column you specify in columns 40 through 43 of the Output sheet. The array elements are put out such that the last element of the named array ends in the column indicated. Note, however, that if all elements in the array cannot be put out on one output record, the array elements must be referenced separately on the Output sheet. Output of individual elements will be discussed later.

The output of an entire array by means of output specifications requires only one specification. An entire array can be written, printed or punched at any time during the run or at end of job, depending on how the output specification is conditioned. Again, during the run, this is possible only if a single output record can contain this entire array.

You must specify how you want the data elements to appear on the output record. Alphameric elements appear on an output record just as they appear in storage; however, numeric array elements may be edited or unedited. If no editing is specified, the elements are printed or punched just as they appear in storage, with the last element ending in the end position column of the output file. In other words, one element will immediately follow another with no punctuation and no blanks between elements.

Usually, printed array output is easier to read and has a better appearance if edit codes or edit words are used to punctuate the data and insert spaces between elements. If punched output, disk output, or tape output is desired, generally the array output records are used as input the next time the program is run. Therefore, editing is usually not specified, so the elements will be in the appropriate format to be used as input.

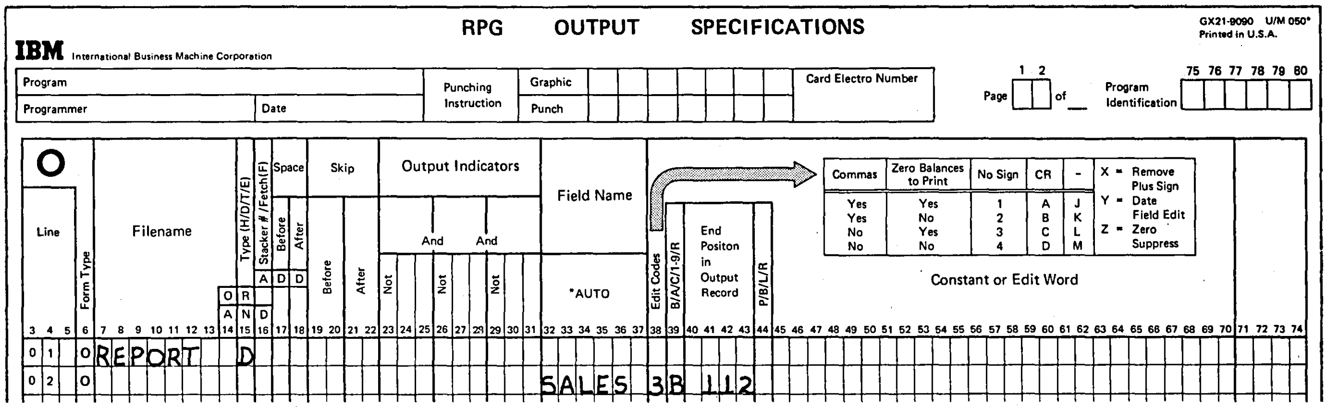
Editing affects the position which is specified as the end position of the output record. If each element in a 5-element array contains seven characters, 35 positions would be necessary to output the entire array in unedited form. On the other hand, if punctuation and blanks are inserted for each element, the number of positions required increases. When specifying an end position, you must allow enough positions to output all edited elements.

Regardless of whether editing is specified or not, when output of an entire array is performed, every element of that array is put out in the same format. If an edit code or edit word is specified, every element is edited in the same way. Since all elements of an array contain the same type of information, ordinarily you want the elements punctuated in the same way. If, however, one element must be edited differently from another element in the same array, you must put out the elements separately. The means of referencing individual elements of an array is discussed later in this section.

When an edit code is specified in column 38 of the Output sheet, every element of the named array will be punctuated accordingly. Furthermore, any edit code specified for an entire array also causes two blank spaces to be inserted between each element. The insertion of blanks is taken into consideration by the program so that the last element ends in the position specified.

As shown in Figure 9-12, the edit code *3* causes all five elements of the SALES array to be printed with decimal points inserted, leading zeros suppressed, and zero balances present. In addition, two blanks are automatically printed before each element since an edit code was specified.

If no edit code specifies exactly how you want the array fields to be edited, you can specify the punctuation by using an edit word (columns 45-70). In this way, you can edit array elements with dollar signs, zero suppression, blanks, constant words, or any combination of punctuation desired. When edit words are used, all punctuation must be specified. Unlike edit codes, edit words do not cause two blanks to be automatically inserted in the output record before each array element. Figure 9-13 shows an edit word specified without blanks; one element of SALES immediately follows the next on the output record. Any extra blanks which are to appear must be indicated in the edit word by an &. Notice, in Figure 9-14 that the two blanks specified are to be printed as part of every element. Thus, the second blank following the last element will be the character which ends in the end position column. Notice that an additional two columns have been allowed for each element (five elements). The end position column has thus been increased by ten over that in Figure 9-13.



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of __ | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Type (H/D/T/E)	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record
0 1	O	REPORT	D				
0 2	O			SALES	3	B	112

Space
Skip
Output Indicators
And
And
Not
Before
After
Stacker #/Fetch(F)
*AUTO
P/B/L/R
Commas | Zero Balances to Print | No Sign | CR | -
Yes | Yes | 1 | A | J
Yes | No | 2 | B | K
No | Yes | 3 | C | L
No | No | 4 | D | M
X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress
Constant or Edit Word

00456	01783	29684	00000	08063

Array

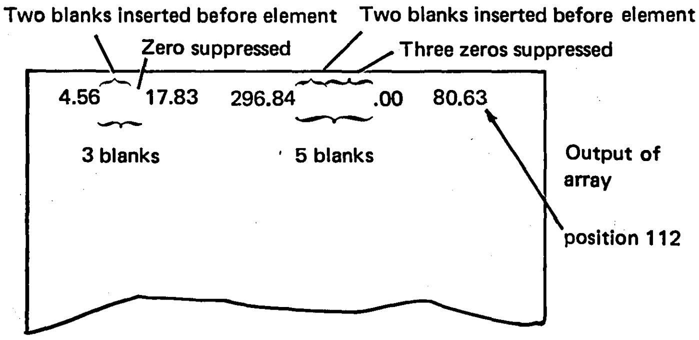
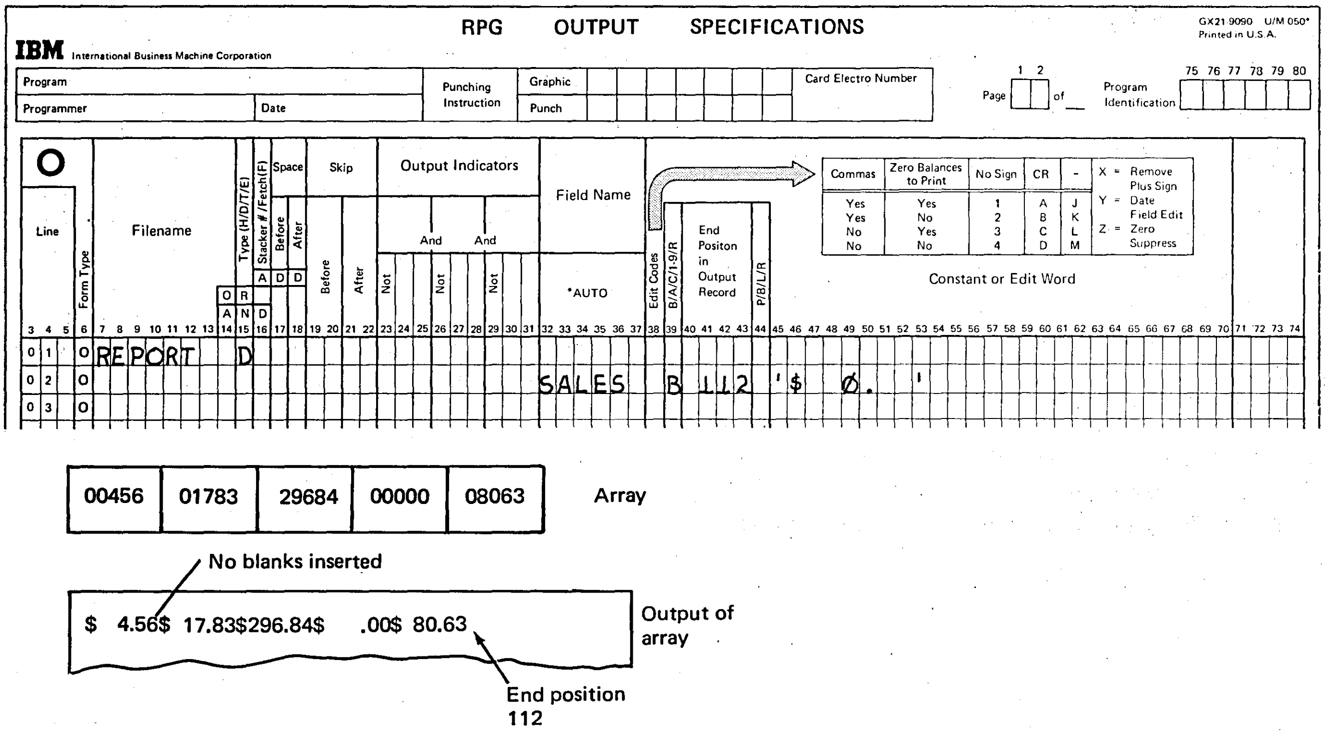


Figure 9-12. Output of an Entire Array With Edit Codes



RPG OUTPUT SPECIFICATIONS

```
01 O REPORT  D
02 O                              SALES  B 112 '$   Ø.   '
03 O
```

00456	01783	29684	00000	08063	Array

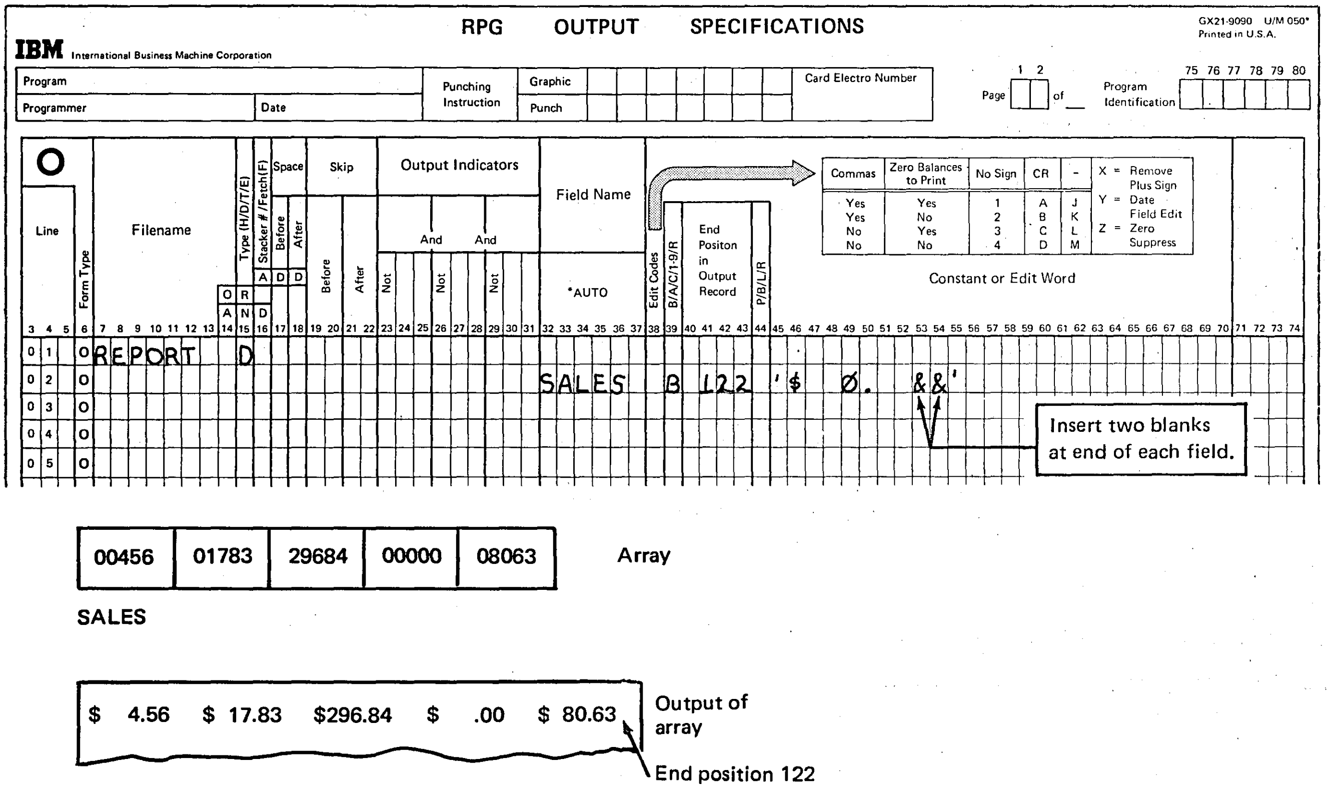
No blanks inserted

```
$  4.56$ 17.83$296.84$    .00$ 80.63
```

Output of array

End position 112

Figure 9-13. Output of an Entire Array With Edit Words



RPG OUTPUT SPECIFICATIONS

```
01 O REPORT  D
02 O                              SALES  B 122 '$   Ø.   &&'
03 O
04 O
05 O
```

Insert two blanks at end of each field.

00456	01783	29684	00000	08063	Array

SALES

```
$  4.56   $ 17.83   $296.84   $    .00   $ 80.63
```

Output of array

End position 122

Figure 9-14. Editing Every Element of an Array

Accumulating Groups of Totals

As you have seen, arrays can be used to accumulate a total. In a previous example, elements containing daily sales were added to obtain monthly totals, which were stored in the MONTH array.

To carry this concept further, one of the most common uses of arrays is accumulating more than one group of totals. Such a procedure is called *rolling of totals,* since one total is used to obtain a greater total, which is then used to calculate an even larger total, and so on. Each total is *rolled into* or accumulated into the next total.

Figure 9-15 shows the organization of the Nelson Company. The company's two regions are each divided into three branches, which are in turn made up of two to six stores each.

Company sales data is recorded on cards as shown in Figure 9-16. For each store there is a separate record providing the 12 sales amounts for each month of the year. The sales records are organized such that stores are grouped within a branch and branches grouped within a region. Three one-position fields on each record identify it with a particular store, a particular branch, and a particular region.

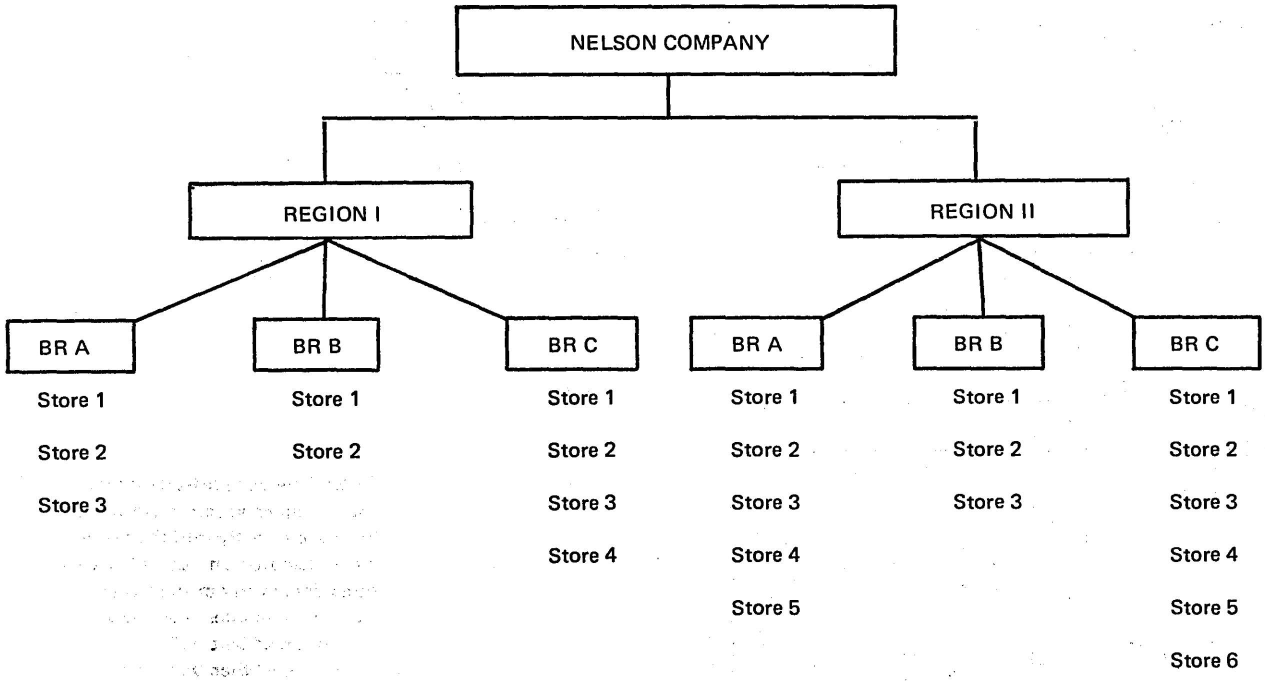
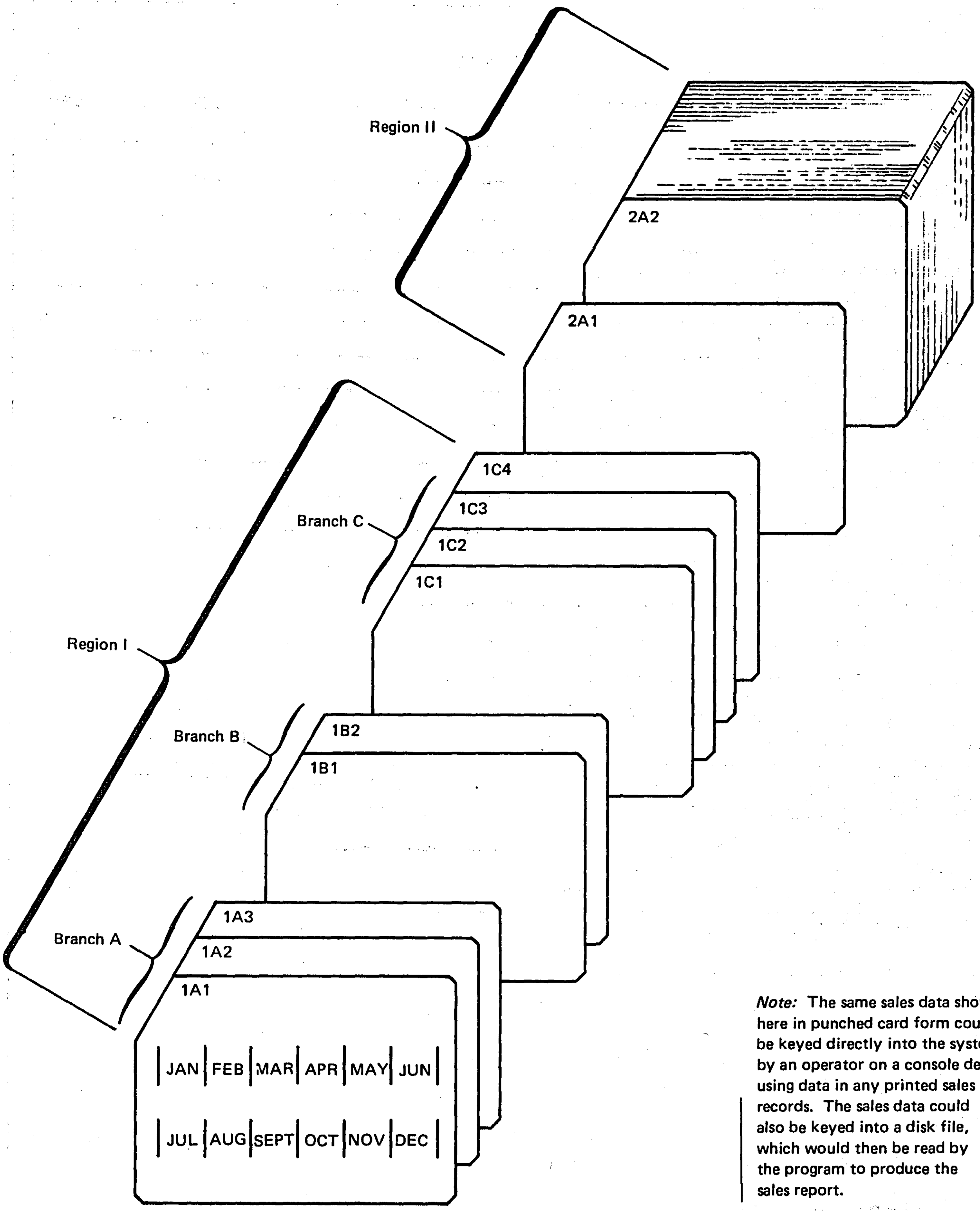
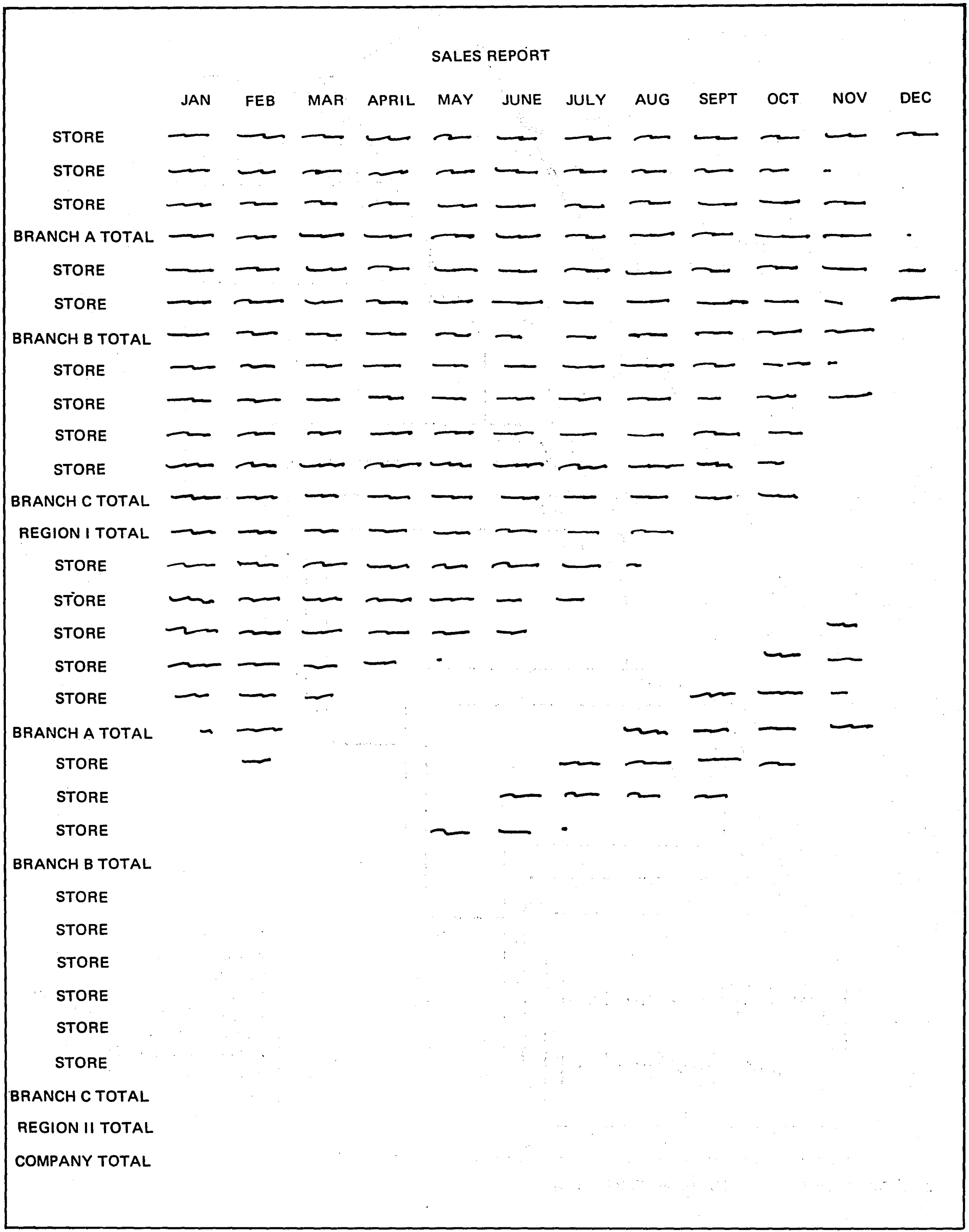


Figure 9-15. Company Organization by Groups



Note: The same sales data shown here in punched card form could be keyed directly into the system by an operator on a console device using data in any printed sales records. The sales data could also be keyed into a disk file, which would then be read by the program to produce the sales report.

Figure 9-16. Sales Records Organized by Groups Within Groups



SALES REPORT

	JAN	FEB	MAR	APRIL	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
STORE												
STORE												
STORE												
BRANCH A TOTAL												
STORE												
STORE												
BRANCH B TOTAL												
STORE												
STORE												
STORE												
STORE												
BRANCH C TOTAL												
REGION I TOTAL												
STORE												
STORE												
STORE												
STORE												
STORE												
BRANCH A TOTAL												
STORE												
STORE												
STORE												
BRANCH B TOTAL												
STORE												
STORE												
STORE												
STORE												
STORE												
STORE												
BRANCH C TOTAL												
REGION II TOTAL												
COMPANY TOTAL												

Figure 9-17. Sales Report by Groups Within Groups

A sales report must be produced showing the monthly sales for each store, for each branch, for all branches within a region, and for both regions (the total monthly sales of the entire company). The report, a series of accumulated totals, should look like the one in Figure 9-17.

To produce the report, four arrays of 12 elements each should be set up, as shown in Figure 9-18. The first array, STR, will be used to hold the 12 sales amounts entered from the sales records. The other three arrays will be used to accumulate the necessary totals for each branch, each region, and the entire company.

In general, this program should accumulate store totals into the BRNCH array, branch totals into the REG array, and region totals into the COMP array. Thus, the specifications must perform two functions:

- Add all elements of one array to all elements of another array.

- Print all elements of each array.

To have the program produce the correct totals, you must specify that one array is to be added to another array and printed. To do this, the two fields which identify a record with a particular branch and region should be specified as control fields. A change in the branch (or region) control fields will cause a control break, indicating the records for all stores in a particular branch (or region) have been processed.

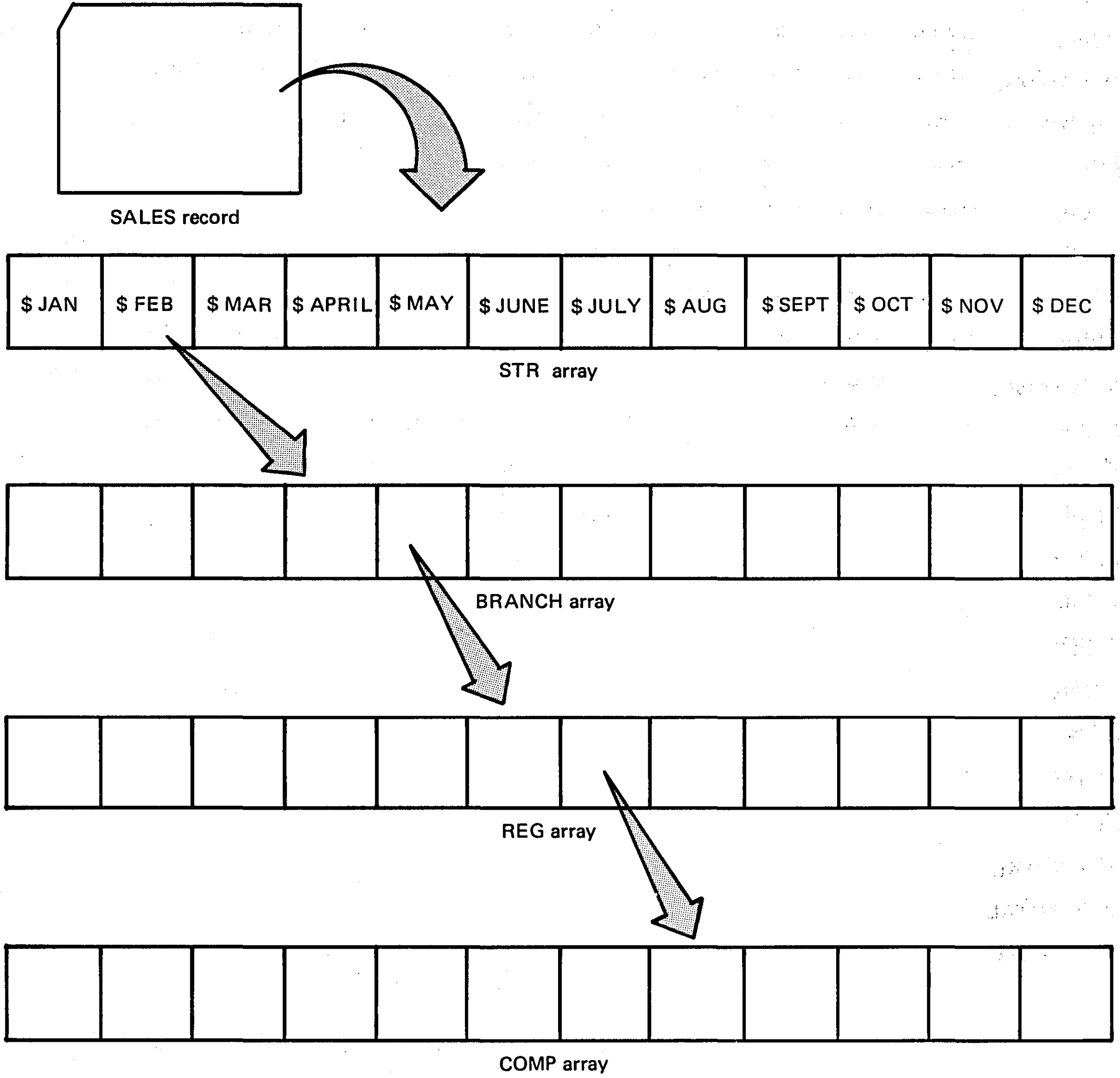


Figure 9-18. Four Arrays for Group Totals

As shown in Figure 9-19, control level indicator L1 is turned on when the first record is read for a store in a different branch. Likewise, L2 is turned on (and thus L1 is automatically turned on) when the first record is read for a store in a different region. The specifications on lines 06-17 merely describe the store sales data for each month of the year.

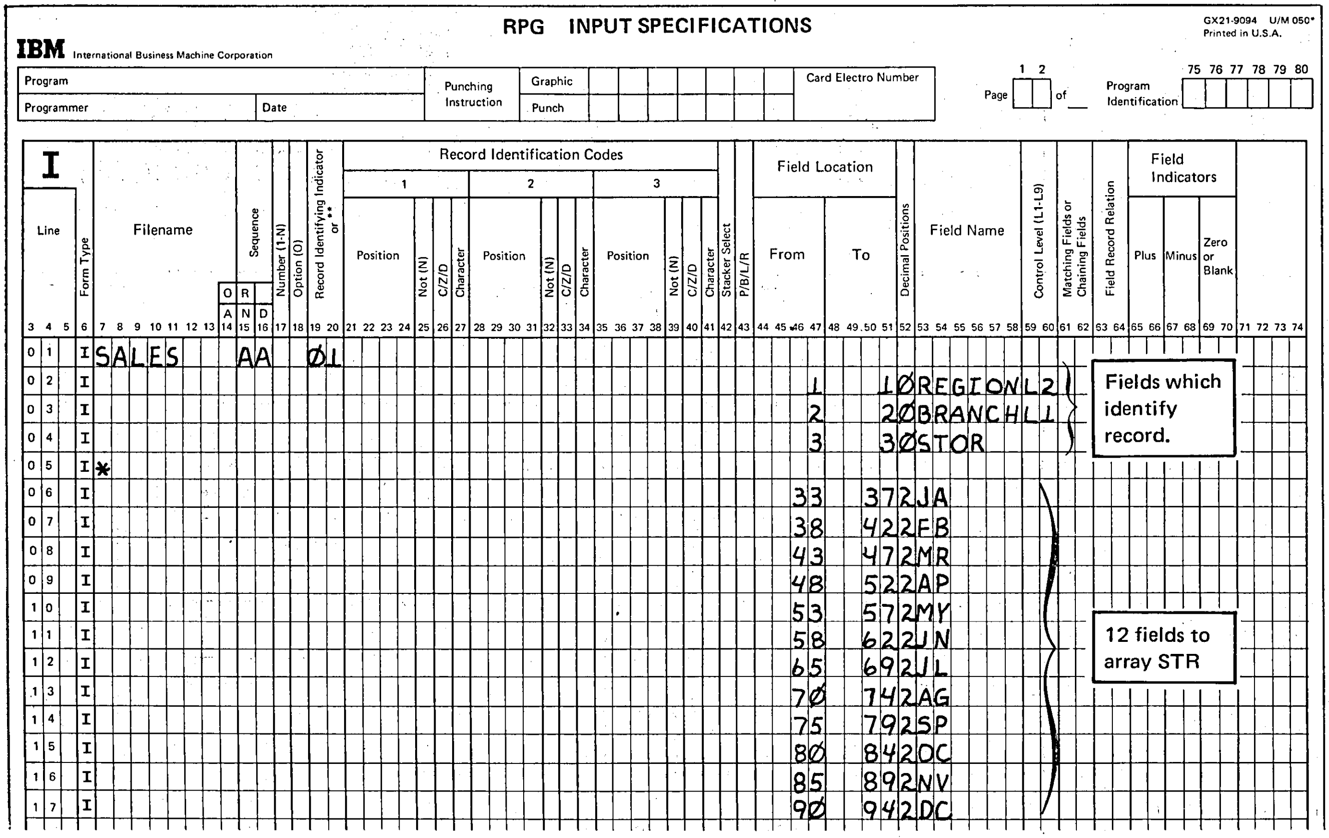
The specifications in Figure 9-20, insert A, illustrate how control level indicators are used to control the performance of calculations and output.

As a sales record is read, the 12 monthly totals for that store are placed in the array STR. (How data gets into an array will be discussed later.) Each time new data is placed in STR (every time a card is processed), the elements are added to the BRNCH array to accumulate totals for the branch (Calculation sheet, line 01). The store totals are printed as each record is processed, because every time a new card is read the data previously in the STR array is replaced by the totals for the next store (output lines 01-02).

When all store records for a particular branch have been read and their totals printed and accumulated in the BRNCH array, an L1 control break occurs. The control break is indicated by reading the first store record in the next branch. Before processing this next record, the branch totals are printed and the BRNCH array is filled with zeros to prepare for accumulating the next branch totals (Figure 9-20, insert B, lines 03-04). Before printing and zeroing the BRNCH array, however, the branch totals are added to the REG array (Calculation sheet, line 02).

The same program cycles are repeated for the rest of the records in region I. Remember, however, that data is accumulated into the REG array only when processing for a branch is complete (L1 on).

Once records for all branches within region I have been processed, L2 is turned on, indicating the first record in the next region has been read, but not yet processed. With L2 on, the 12 accumulated region totals are printed (Output sheet, lines 05-06). Before output of the REG array, however, calculations conditioned by L2 on are performed; that is, the region totals are added to the company array COMP (Calculation sheet, line 03). The calculation is done before output so the region totals can be saved before REG is filled with zeros.



IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	From	To	Decimal Positions	Field Name	Control Level (L1-L9)
01	I	SALES	AA			Ø1					
02	I						1	1	Ø	REGION	L2
03	I						2	2	Ø	BRANCH	L1
04	I						3	3	Ø	STOR	
05	I	*									
06	I						33	37	2	JA	
07	I						38	42	2	FB	
08	I						43	47	2	MR	
09	I						48	52	2	AP	
10	I						53	57	2	MY	
11	I						58	62	2	JN	
12	I						65	69	2	JL	
13	I						7Ø	74	2	AG	
14	I						75	79	2	SP	
15	I						8Ø	84	2	OC	
16	I						85	89	2	NV	
17	I						9Ø	94	2	DC	

Fields which identify record.

12 fields to array STR

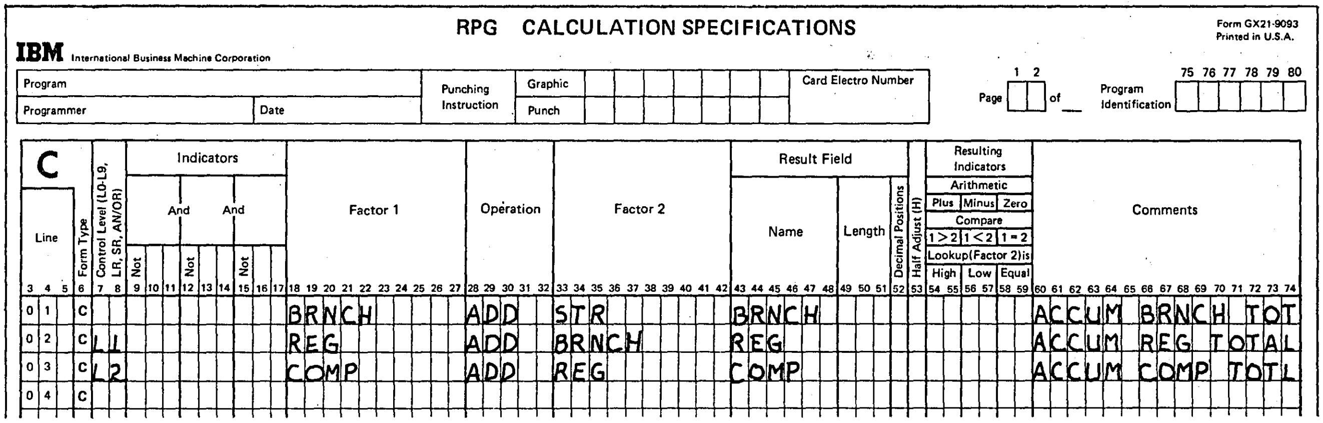
Figure 9-19. Identifying Groups by Assigning Control Level Indicators

The same procedure is followed for all store records in region II. During every program cycle, the store totals are printed and accumulated to form a branch total; when L1 is on, the branch total is printed and accumulated into a region total.

When the end of file is reached, the LR indicator is turned on. Automatically, all control level indicators assigned (L1 and L2) are also turned on. Therefore, after the last store record has been printed and the store totals added to BRNCH (Figure 9-20, insert A, line 01), any specifications conditioned by L1, L2, or LR are performed. In other words, the totals for the last branch are added to REG (Figure 9-20, insert A); then the region totals are added to COMP. Following the calculations, three sets of totals are printed; totals for the last branch (Figure 9-20, insert B, lines 03-01); then the region II totals; and, finally, the company totals.

REFERENCING INDIVIDUAL ELEMENTS OF AN ARRAY

In addition to referencing all elements of an array, you can use an individual array element in calculations or output. Suppose you have an array with each element containing the quantity in stock of a particular part manufactured by your company. Element 1 contains the quantity in stock for part #1, element 2 for part #2, and so on. When a shipment of ordered parts is received, the quantity in stock must be updated to reflect the current inventory. This means you should reference (add to) only particular elements of the array.



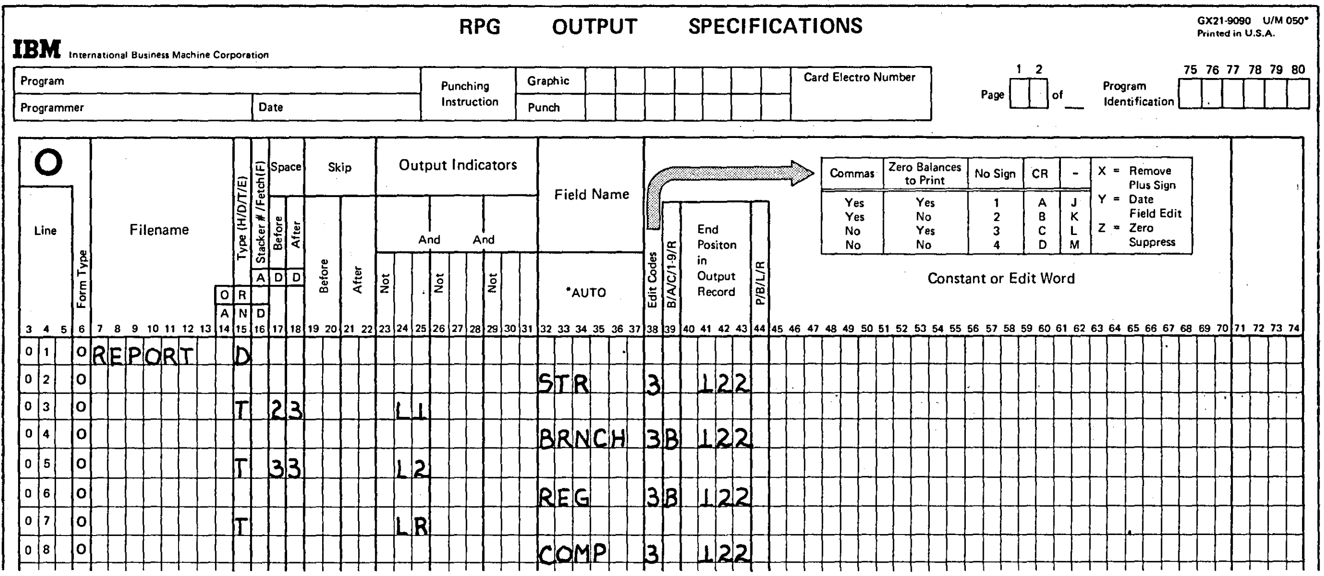
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C			BRNCH	ADD	STR	BRNCH					ACCUM BRNCH TOT
02	C	L1		REG	ADD	BRNCH	REG					ACCUM REG TOTAL
03	C	L2		COMP	ADD	REG	COMP					ACCUM COMP TOTL
04	C											



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	REPORT	D											
02	O								STR	3		122		
03	O		T	2	3			L1						
04	O								BRNCH	3	B	122		
05	O		T	3	3			L2						
06	O								REG	3	B	122		
07	O		T					LR						
08	O								COMP	3		122		

Figure 9-20. Accumulation and Output of Group Totals Using Arrays

Indexing an Array

As you learned, if a calculation or output specification contains an array name alone, that specification is automatically performed for every element of the array. To reference only a single element of an array, you must identify that element for the RPG II program. This is done by placing a comma after the array name, followed by an index which points to the particular element (Figure 9-21). This *index* can be either the actual number of the element to be referenced or the name of a field containing the number of the element to be used.

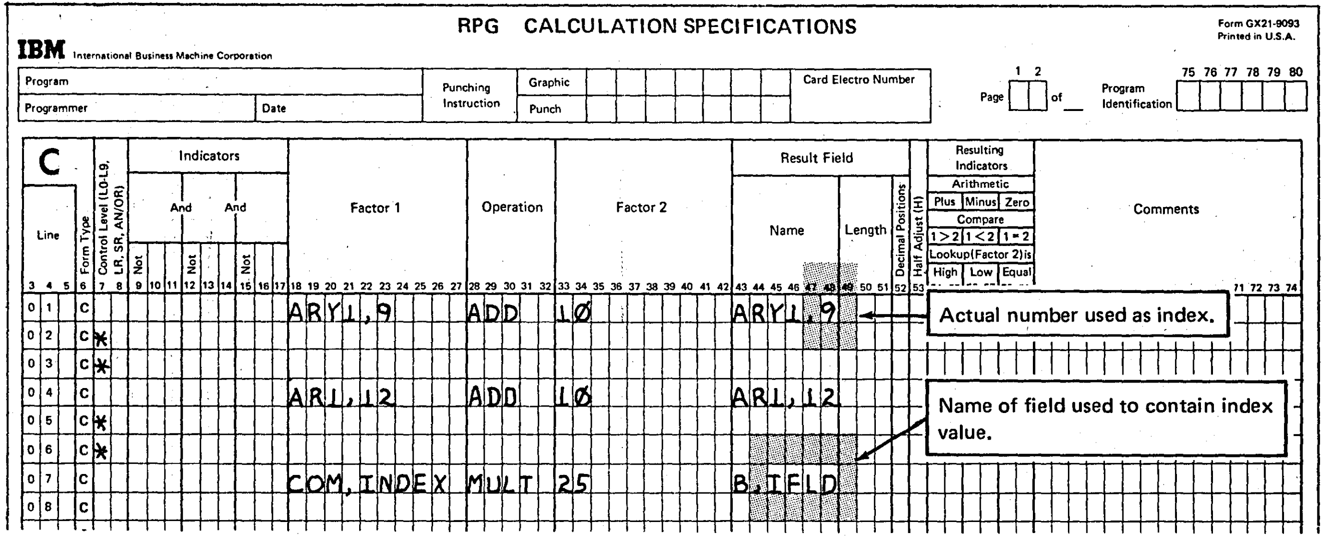
Recall *Defining an Array.* The name used to refer to an array cannot exceed six characters in length. When referencing individual fields, both the array name and an index are necessary to refer to the data. Therefore, usually the array name, plus the comma, plus the index cannot exceed six characters. The name used to refer to an individual array element must be one to six characters long unless the name (with index) used to refer to an individual array element is specified *only* as Factor 1 or Factor 2 on the Calculation sheet. In this case, the array name plus comma plus index may be as long as ten characters. However, the array name portion of the reference still cannot exceed six characters.

Figure 9-21, line 01, shows a valid reference to the ninth element of an array named ARY1. However, if the array contains ten or more elements, some of which may have to be referenced, the name of this array would have to be shortened to provide enough positions for the index (Figure 9-21, line 04). The limit of six characters applies even if the name of a field is used as an index. As line 07 shows, if an index field IFLD is specified, only one character (B) can be used as the name of the array because the indexed name is specified under Result Field. However, COM, INDEX on line 07 is valid, even though longer than six characters, because it is specified only under factor columns.

Specifying an Index Which Does Not Change

If you know exactly which element is to be used in a calculation or output operation and the specification is to reference the same element in every program cycle, you may use a constant as the index. Assume a 7-element array (SLS) is defined to contain a salesman's six daily commission amounts and his total commission for the week. The six daily amounts from one of the salesmen's input records are read into elements 1-6 of the array. The seventh field on the input record contains zeros and is read into element 7 of the array (Figure 9-22, insert A).

The array elements are defined as 5-digit numbers with two decimal positions. Once the data is in the array, the XFOOT calculation operation is performed to add all elements of the array and place the total in the seventh element (Figure 9-22, insert B). The weekly total for every salesman is always stored in the seventh element. Therefore, the actual number 7 can be specified as the index. In addition, a $25 bonus is to be added to a salesman's total if his weekly commission exceeds $175 (Figure 9-22, insert C). Thus, in every program cycle, element 7 must first be compared to $175 to determine if the bonus is to be added to the contents of element 7.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program ____ Programmer ____ Date ____

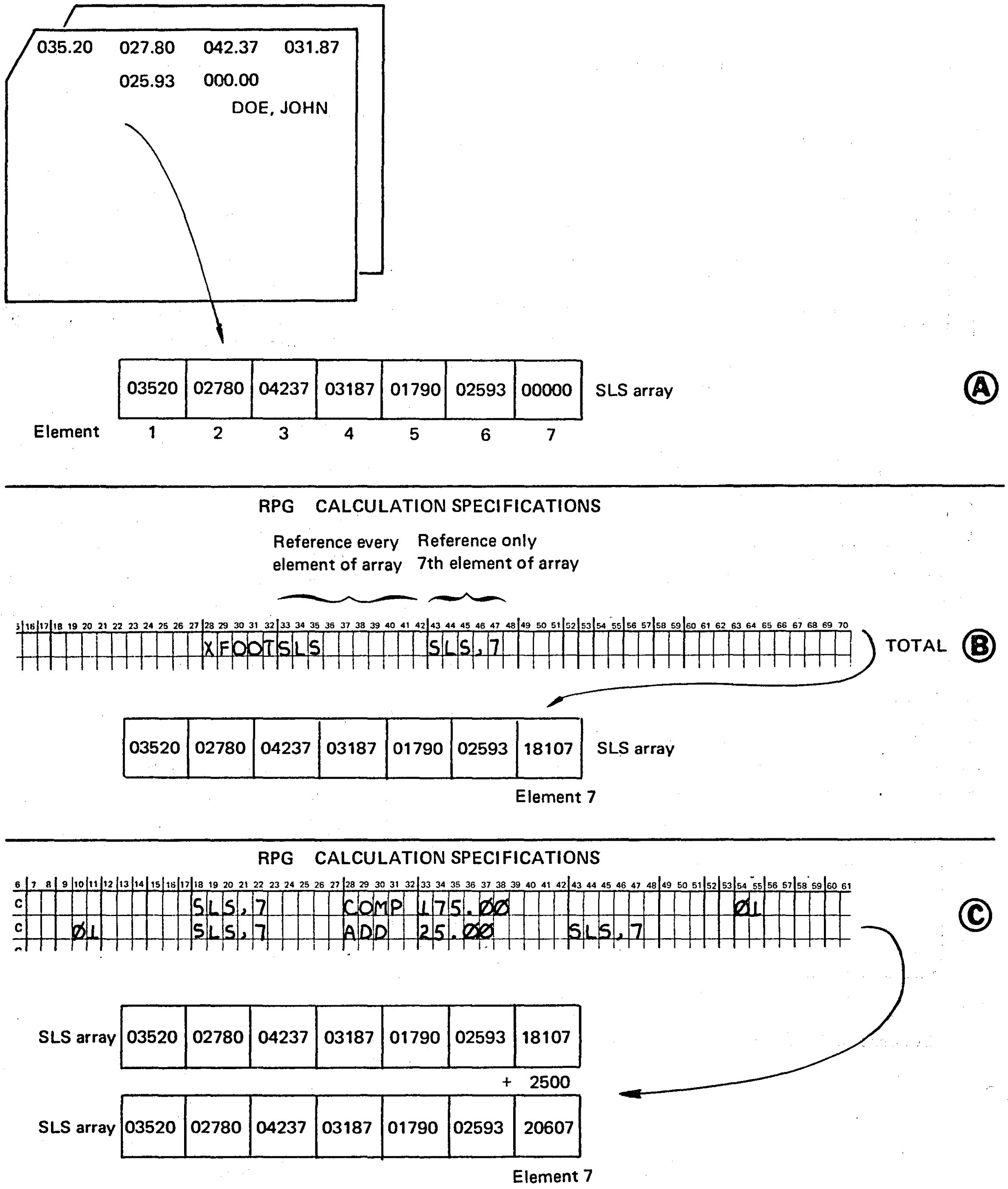
Punching Instruction: Graphic ____ Punch ____

Card Electro Number ____

Page __ of __ Program Identification ____

Line	Form Type	Control Level / Not	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Comments
01	C			ARY1,9	ADD	1Ø	ARY1,9		Actual number used as index.
02	C	*							
03	C	*							
04	C			AR1,12	ADD	1Ø	AR1,12		
05	C	*							Name of field used to contain index value.
06	C	*							
07	C			COM,INDEX	MULT	25	B,IFLD		
08	C								

Figure 9-21. Referencing a Particular Element of an Array


035.20 027.80 042.37 031.87
025.93 000.00
DOE, JOHN
03520 02780 04237 03187 01790 02593 00000 SLS array
Element 1 2 3 4 5 6 7
A
RPG CALCULATION SPECIFICATIONS
Reference every element of array
Reference only 7th element of array
TOTAL
B
03520 02780 04237 03187 01790 02593 18107 SLS array
Element 7
RPG CALCULATION SPECIFICATIONS
C
SLS array 03520 02780 04237 03187 01790 02593 18107
+ 2500
SLS array 03520 02780 04237 03187 01790 02593 20607
Element 7

```
          XFOOTSLS       SLS,7
```

```
C          SLS,7     COMP 175.ØØ                  Ø1
C  Ø1      SLS,7     ADD  25.ØØ      SLS,7
```

Figure 9-22. Specifying a Number as an Index

]

Specifying an Index Which Can Be Changed

On the other hand, if the array element will vary when a particular specification is performed, the index should be a field name rather than an actual number. In this way, the number stored in the index field can be changed during the program to indicate which array element is to be referenced.

An array (STK) is used to contain the quantities in stock of all parts manufactured by a company. Element 1 of the array contains the quantity for part #1, element 2 for part #2, and so on. When additional parts are manufactured, the values in the appropriate elements must be updated. Therefore, records are punched daily for each type of part produced. Each record contains the part number (NM) and the quantity of that part produced (QTY).

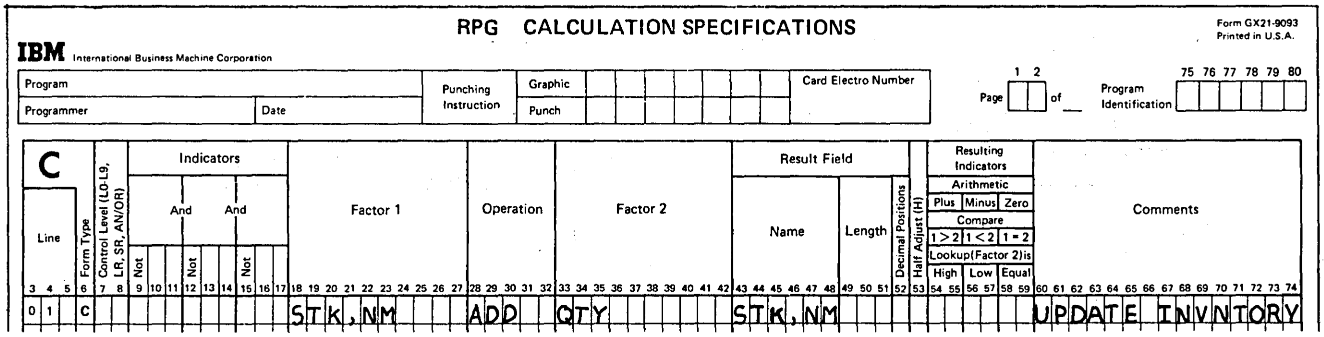
To perform the updating, the contents of the QTY field must be added to one of the array elements for every record processed. Thus, an index must be used in order to reference only the individual element to be updated. Since each daily record is for a different part number, the array element to be increased will vary each time the specification is performed. For this reason, an actual number cannot be specified as the index, because QTY would be added to the same element for every part number. Instead, the NM field, which contains the part number for each record, can be specified as the index (Figure 9-23). Then, every time the addition specification is performed, the part number just stored in NM indicates which element of the array is to be referenced.

Output of Individual Elements of an Array

To put out individual elements of an array, you code the same output specifications you would for normal fields. The only difference is that under Field Name on the Output sheet you must specify the array name followed by a comma and an index. The index then points to the particular element to be put out (Figure 9-24).

Thus, referencing individual array elements for output is the same as referencing them for calculations. If the same element is to be put out every time the output specification is performed, an actual number can be used as an index. Otherwise, if different elements are to be put out individually, a field should be specified which contains the changing index value. In any case, the array element (array name plus comma plus index) on the Output sheet cannot exceed six characters in length.

Edit codes and edit words can be used to punctuate an individual numeric array element. If an entire array is to be put out but the elements require different punctuation, each element and its editing should be specified individually. Editing to be done on an individual array element is specified and performed just as it would be for any normal element. This means that, if an edit code is specified for an individual array element, two blanks are *not* automatically inserted before the element, as was the case with an entire array. Furthermore, although any type of output can be edited, editing is generally not specified for an array element which is to be punched on a card or written on disk to be used as input to another run.



IBM International Business Machine Corporation

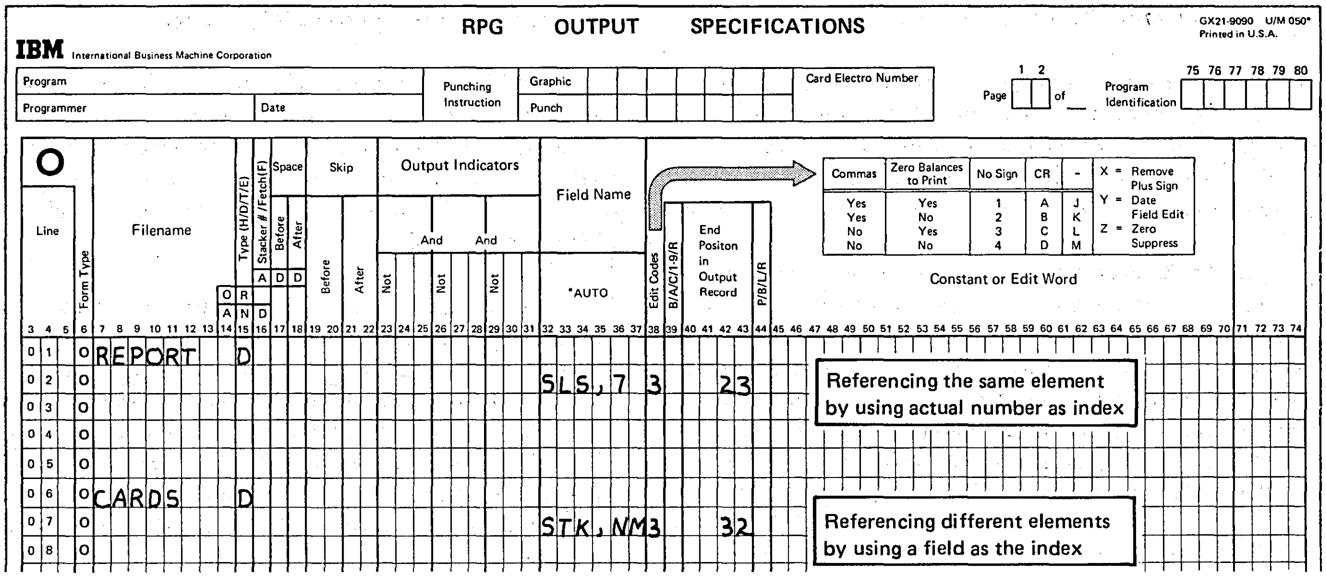
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page of
Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C			STK,NM	ADD	QTY	STK,NM					UPDATE INVNTORY

Figure 9-23. Specifying the Name of a Field as an Index



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
0 1	O	REPORT	D				
0 2	O			SLS,7	3	23	
0 3	O						
0 4	O						
0 5	O						
0 6	O	CARDS	D				
0 7	O			STK,NM	3	32	
0 8	O						

Referencing the same element by using actual number as index

Referencing different elements by using a field as the index

Figure 9-24. Output of Individual Array Elements

Referencing Only Part of a Field

When a field is referenced in a specification, all characters within that field are used in the calculation or output. However, you may wish to reference only some of the data stored in a field. For example, consider the case during address printing where the zip code is within the same field as the city and state on an input record but must be printed on a separate line on the output record (Figure 9-25).

The indexing capability of arrays can be used to enable you to reference specific characters from an input field. This is accomplished by setting up two arrays; one to contain the entire field of data and one to hold only the specific characters you want to reference.

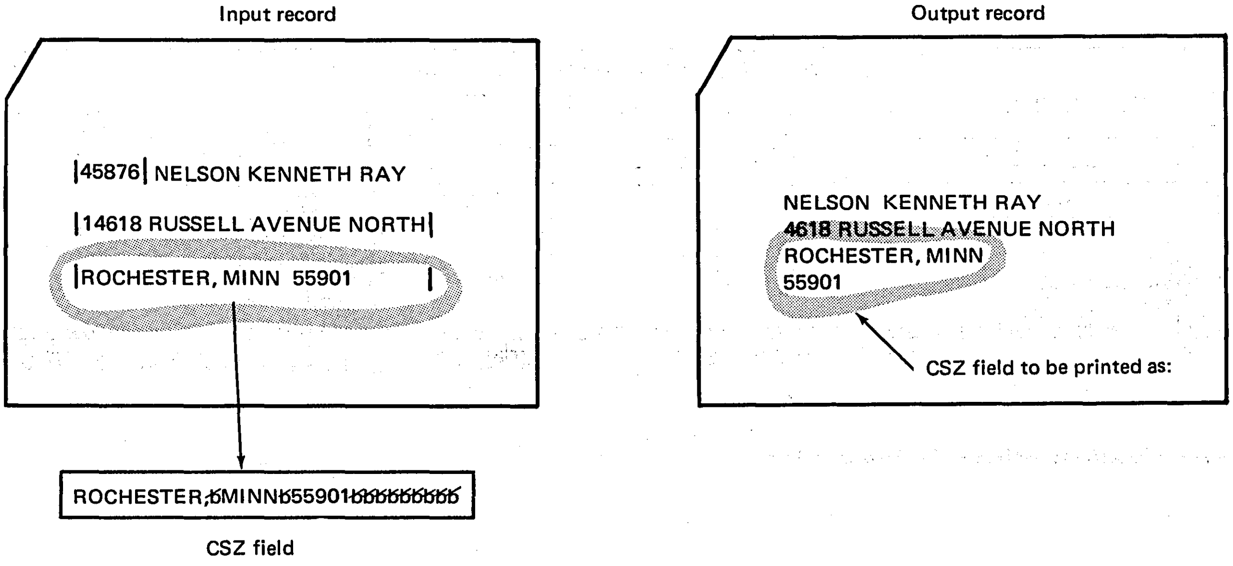

Input record
|45876| NELSON KENNETH RAY
|14618 RUSSELL AVENUE NORTH|
|ROCHESTER, MINN 55901 |
ROCHESTER,bMINNb55901bbbbbbbbb
CSZ field
Output record
NELSON KENNETH RAY
4618 RUSSELL AVENUE NORTH
ROCHESTER, MINN
55901
CSZ field to be printed as:

Figure 9-25. Referencing Parts of a Field Separately

First, the entire field from which you wish to use data is stored in an array of the same name as shown in Figure 9-26. (See *Loading Arrays, Storing Input Data into Execution Time Arrays* later in this chapter for an explanation of this method of loading an array.) This array is previously defined as containing as many one-byte elements as there are characters in the field to be referenced. Thus, each character of the one field is actually stored in a separate element of the array. The array elements can then be referenced one at a time (using an index) until an element containing a specific character is located. This process of checking the elements of an array for particular data is referred to as *field scanning.*

After scanning the elements and locating a specific character, you can then move that character and any characters (elements) on either side of it to a smaller array. This array will then contain the portion of the original input field which you wish to reference separately in calculations or output.

For an address printing program, let's assume the input records are defined as shown in Figure 9-27. The CSZ field contains the city/state and zip code. Although names of the city and state may vary in length, the zip code is always five digits long. Any righthand, unused positions of the CSZ field will contain blanks.

The two arrays for this program are defined with the extension specifications in Figure 9-28. CSZ is set up to contain 30 elements, one for each character of the CSZ field from the input record. The five elements of the ZIP array will be used to contain the zip code portion of the CSZ field.

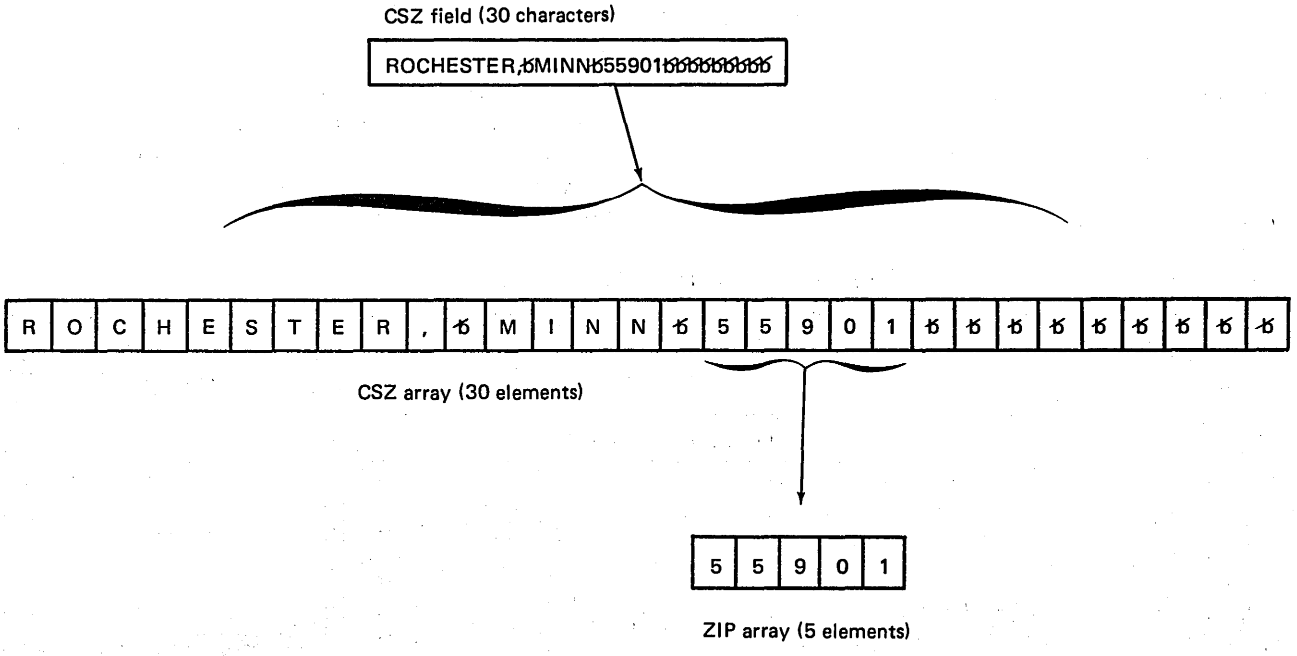
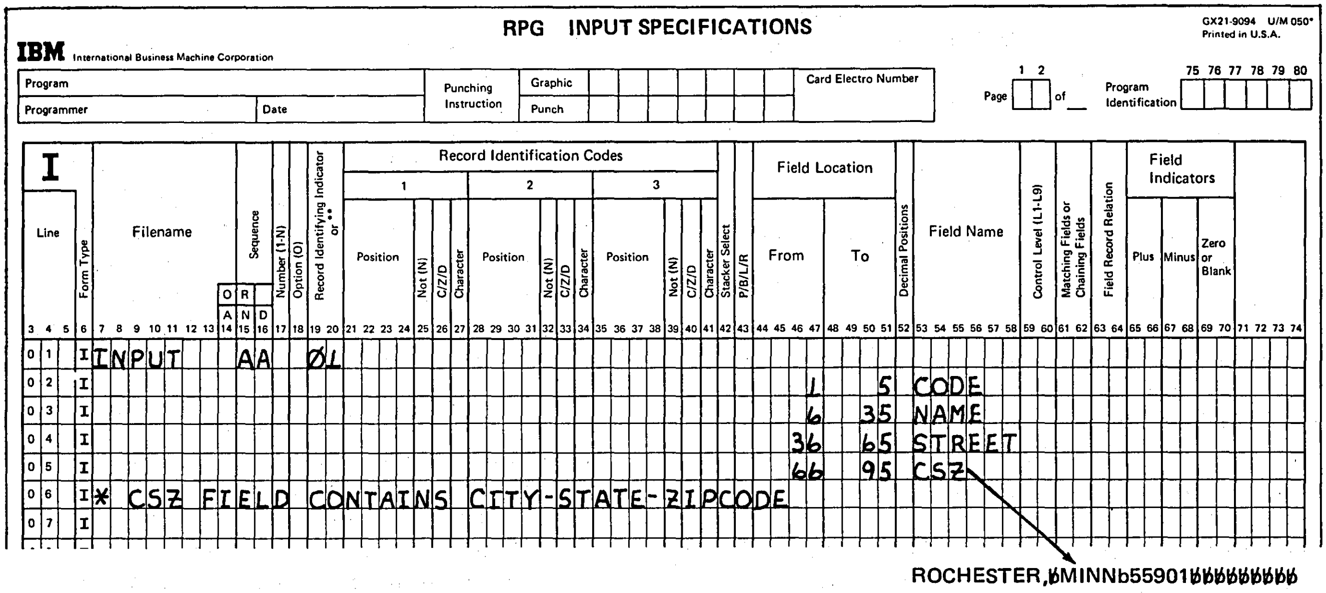

CSZ field (30 characters)
ROCHESTER,ƀMINNƀ55901ƀƀƀƀƀƀƀƀƀ
R O C H E S T E R , ƀ M I N N ƀ 5 5 9 0 1 ƀ ƀ ƀ ƀ ƀ ƀ ƀ ƀ ƀ
CSZ array (30 elements)
5 5 9 0 1
ZIP array (5 elements)

Figure 9-26. Isolating Part of a Field



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

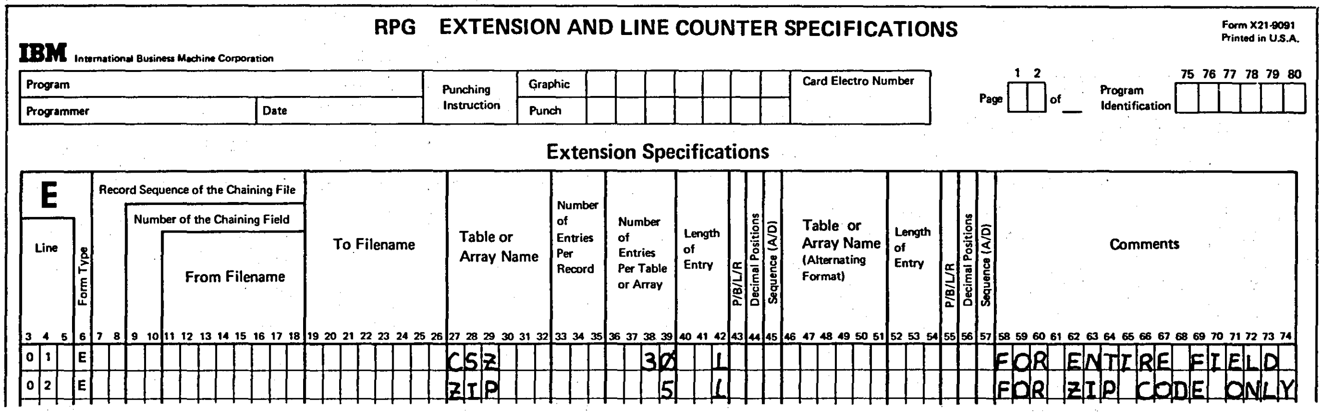
Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page □□ of __ | Program Identification □□□□□□

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Field Location From	Field Location To	Field Name
01	I	INPUT	AA	ØL			
02	I				1	5	CODE
03	I				6	35	NAME
04	I				36	65	STREET
05	I				66	95	CSZ
06	I	* CSZ FIELD CONTAINS CITY-STATE-ZIPCODE					
07	I						

ROCHESTER,ƀMINNb55901ƀƀƀƀƀƀƀƀƀ

CSZ field

Figure 9-27. Defining a Field to be Scanned



RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

IBM International Business Machine Corporation

Form X21-9091 Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page □□ of __ | Program Identification □□□□□□

Extension Specifications

Line	Form Type	Table or Array Name	Number of Entries Per Table or Array	Length of Entry	Comments
01	E	CSZ	3Ø	1	FOR ENTIRE FIELD
02	E	ZIP	5	1	FOR ZIP CODE ONLY

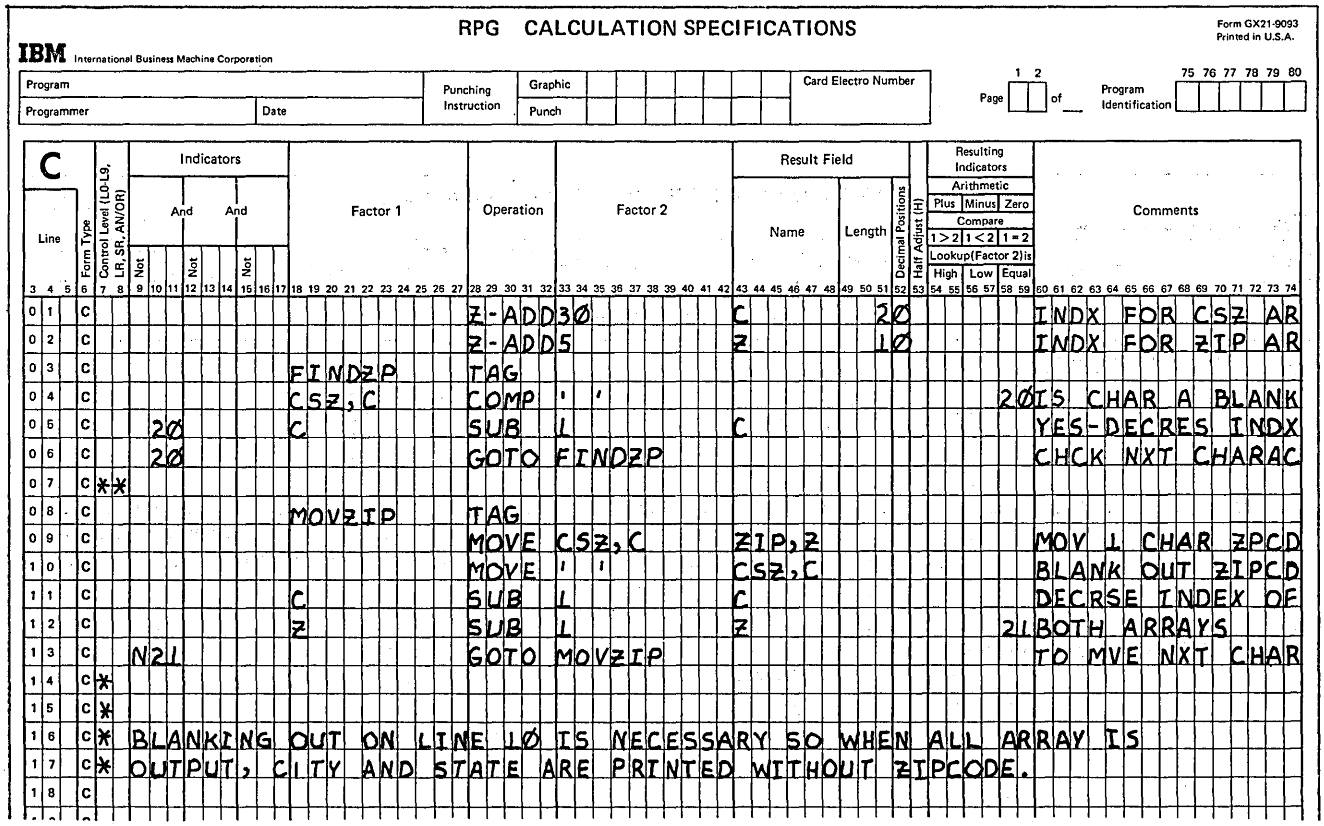
Figure 9-28. Defining Arrays for Field Scanning

To locate the zip code in the CSZ array, the elements must be scanned one at a time, beginning with the last (rightmost) element of the array. Thus, the index field (C) for referencing the individual fields of CSZ is initially set up in the Calculation sheet to contain the value 30 (Figure 9-29, line 01). When the last (rightmost) character of the zip code is located, it should be moved to the rightmost (fifth) element of the ZIP array. Therefore, a 5 is initially set up in the index field (Z) which will reference a particular field of ZIP (Figure 9-29, line 02).

With the index fields set up, the program can begin scanning CSZ for the zip code. The elements of CSZ are checked, from right to left, until the first nonblank character is located. As line 04 shows, a character is compared to a blank. If it is blank, the index value is decreased (line 05) so the next character to the left can be compared to a blank. When one of the characters checked is not a blank (indicator 20 off), the last character of the zip code has been located. This ends the field scanning.

The program can now proceed to perform the next group of calculations which move the located character to the rightmost position of the ZIP array (line 09). The character in the CSZ array which was moved to the ZIP array is now made blank (line 10) so the city and state line can be printed without the zip code. The index values for both arrays are decreased by *1*, so the next zip code character (to the left of the last one moved) can be moved from CSZ to the next portion in the ZIP array. When the value of Z becomes 0 (indicator 21 is on), all five characters of the zip code have been moved to the ZIP array and made blank in the CSZ array.

After calculations, the output specifications in Figure 9-30 cause the name and address to be printed. The NAME and STREET fields are printed exactly as they appear on the input record. City and state, on the other hand, are printed from the CSZ array rather than the input field, because the zip code has been blanked out. The zip code, which was moved to the ZIP array, is then printed alone on the next output line.



IBM International Business Machine Corporation

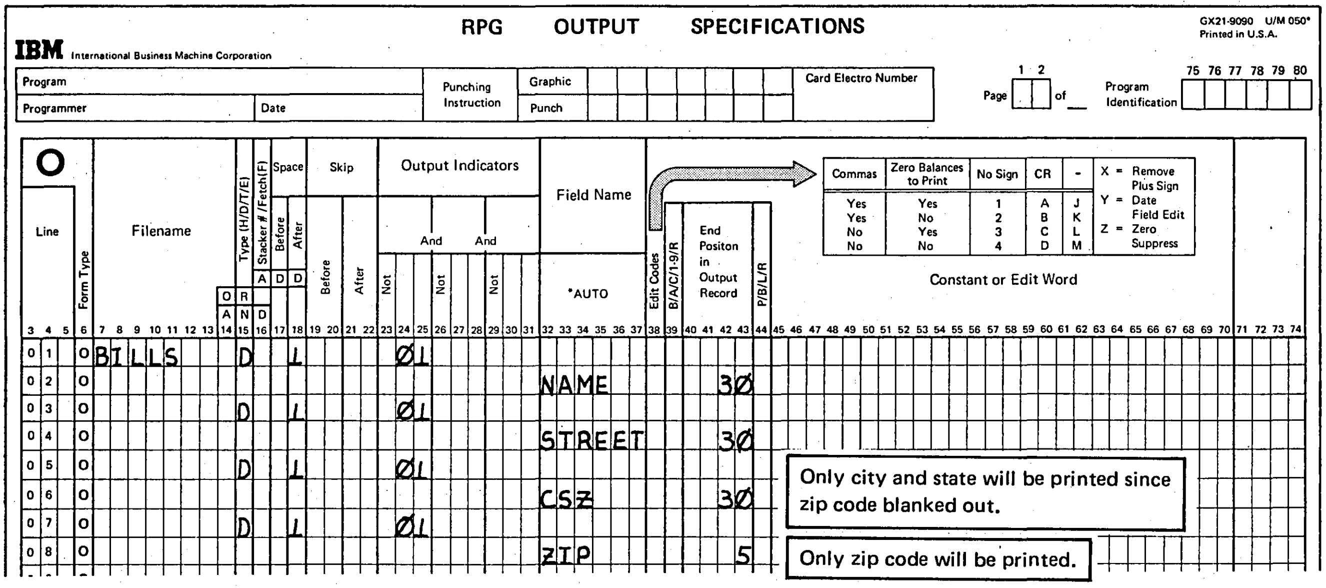
RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page __ of __ | Program Identification

```
Line  Form Type  Control Level  Indicators  Factor 1  Operation  Factor 2  Result Field (Name, Length, Decimal Positions)  Half Adjust  Resulting Indicators  Comments
3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... 27 28 ... 32 33 ... 42 43 ... 48 49 50 51 52 53 54 ... 59 60 ... 74

01 C                         Z-ADD3Ø        C       2Ø       INDX FOR CSZ AR
02 C                         Z-ADD5         Z       1Ø       INDX FOR ZIP AR
03 C           FINDZP    TAG
04 C           CSZ,C     COMP ' '                          2ØIS CHAR A BLANK
05 C       2Ø  C         SUB  1         C                    YES-DECRES INDX
06 C       2Ø            GOTO FINDZP                         CHCK NXT CHARAC
07 C**
08 C           MOVZIP    TAG
09 C                     MOVE CSZ,C     ZIP,Z                MOV 1 CHAR ZPCD
10 C                     MOVE ' '       CSZ,C                BLANK OUT ZIPCD
11 C           C         SUB  1         C                    DECRSE INDEX OF
12 C           Z         SUB  1         Z                  21BOTH ARRAYS
13 C  N21                GOTO MOVZIP                         TO MVE NXT CHAR
14 C*
15 C*
16 C* BLANKING OUT ON LINE 1Ø IS NECESSARY SO WHEN ALL ARRAY IS
17 C* OUTPUT, CITY AND STATE ARE PRINTED WITHOUT ZIPCODE.
18 C
```

Figure 9-29. Field Scanning



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

```
Line  Form  Filename  Type  Space  Output      Field Name  End
      Type                  After  Indicators              Position
0 1   O     BILLS     D     1      Ø1
0 2   O                                        NAME        3Ø
0 3   O               D     1      Ø1
0 4   O                                        STREET      3Ø
0 5   O               D     1      Ø1
0 6   O                                        CSZ         3Ø
0 7   O               D     1      Ø1
0 8   O                                        ZIP          5
```

Only city and state will be printed since zip code blanked out.

Only zip code will be printed.

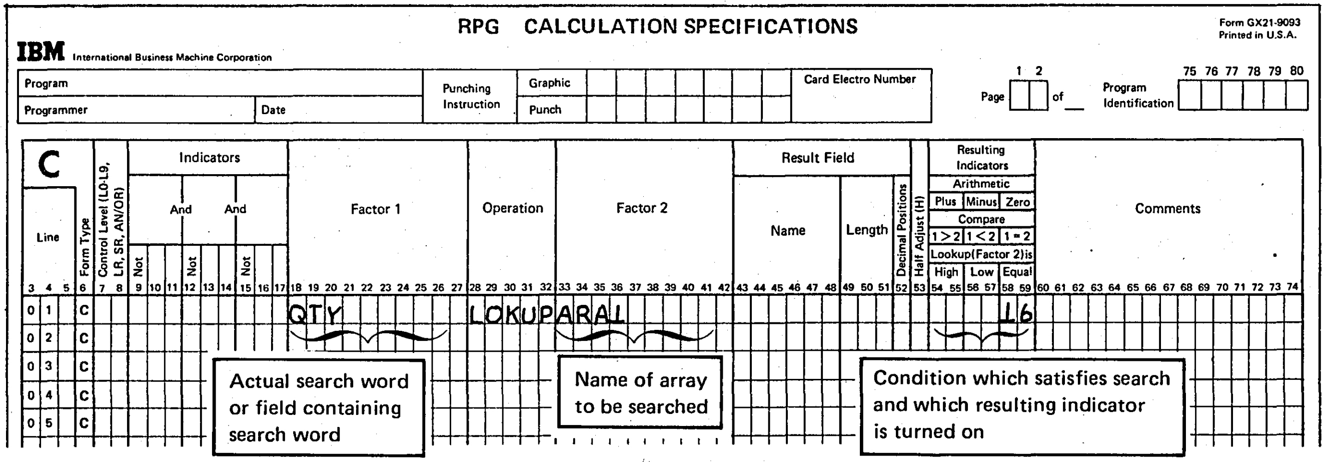
Figure 9-30. Output of Part of a Field

LOKUP OF AN ARRAY

Searching an Array for a Particular Element

An array can be searched to determine if a particular element of data is stored in the array. Actually, the array lookup is coded and performed in almost the same way as a single table lookup. As the Calculation sheet in Figure 9-31 shows, you specify:

1. The search word to be used.
2. The LOKUP operation code.
3. The array to be searched.
4. The condition which must be satisfied.
5. The resulting indicator which turns on if the condition is met.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

```
Line  Form  Factor 1  Operation  Factor 2   Resulting Indicators
      Type                                  Equal (58-59)
0 1   C     QTY       LOKUP      ARAL       16
0 2   C
0 3   C
0 4   C
0 5   C
```

Actual search word or field containing search word

Name of array to be searched

Condition which satisfies search and which resulting indicator is turned on

Figure 9-31. Searching an Array for a Particular Element

The array lookup continues, one element at a time, until the search condition is satisfied or the end of the array is reached, whichever occurs first. As is the case for table lookups, array elements must be in sequence (A or D) if searching for either a low or high condition. Additional coding is necessary if searching an out-of-sequence array for either high or low. (See *Searching an Array for More Than One Element* and *Output During an Array Search.)*

Although array and table searches are similar, there is an important difference you must be aware of. Remember, the array lookup is similar to a single-table lookup, not a two-table lookup. Only one array is specified in the lookup operation. Any element which is referenced as the result of a successful search can only be from the array actually searched. In other words, the array can not be searched to make an element from another array available, as is the case when two related tables are used in a lookup operation. For this reason, no result field may be specified in an array lookup operation.

Starting the Search at a Particular Element

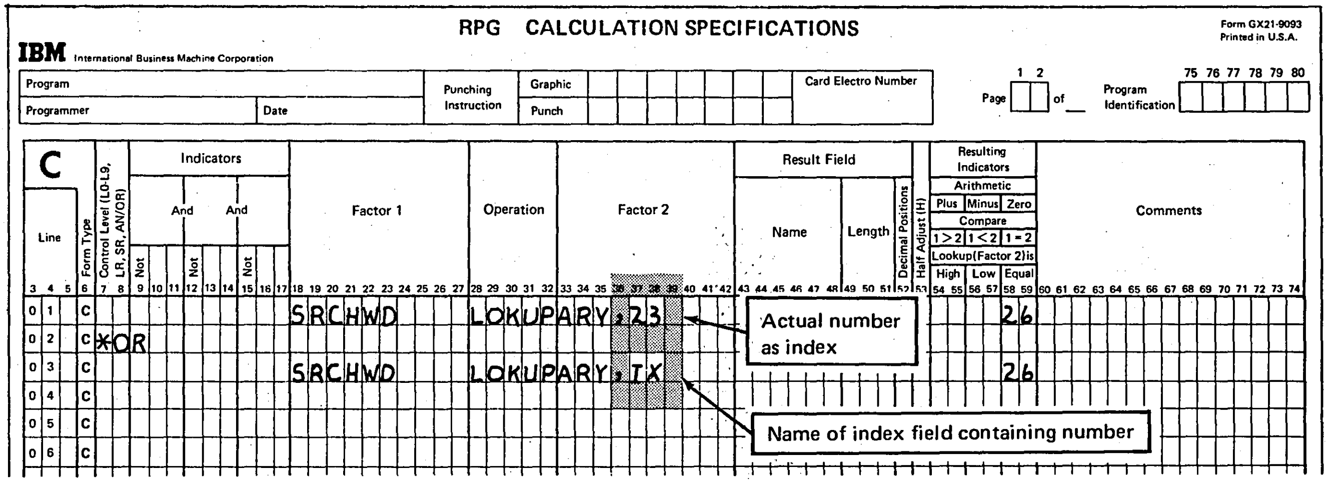
Another very important difference between tables and arrays concerns where the search can begin. In a table search, only the name of the table to be searched can be specified as Factor 2 of the lookup operation. As a result, a table search always begins at the first table element. Likewise, if only an array name is specified as Factor 2 of a lookup operation, the search automatically begins at the first element of the named array.

With arrays, however, you also have the capability of beginning an array search at any element you specify. Under Factor 2, you specify the array name, followed by a comma and an index. The index, whether an actual number or the name of a field containing a number, points to the array element where the search is to begin (Figure 9-32).

In a large array where you know that the value you are searching for is not in a particular section of the array, search time can be greatly decreased by beginning the lookup at a particular element. Suppose you have a 300-element array name ARY containing the values 001 through 300 in ascending sequence. To locate a value of 047, only 47 elements would have to be checked before the search condition was satisfied. However, to locate the value 289, 289 elements would have to be checked, *if* the search began at the first array element.

Now, divide the array into three parts of 100 elements each:

Elements 1-100:	values 001-100.
Elements 101-200:	values 101-200.
Elements 201-300:	values 201-300.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page □□ of __
Program Identification □□□□□□

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High	Low	Equal	Comments
01	C			SRCHWD	LOKUP	ARY,23							26	
02	C	*OR												
03	C			SRCHWD	LOKUP	ARY,IX							26	
04	C													
05	C													
06	C													

Figure 9-32. Starting an Array Search at a Particular Element

For any value of less than 101, the first third of the array is searched, beginning at element 1. For values greater than 100, but less than 201, the second third of the array is searched, beginning at element 101. Likewise, a search is started at element 201 to locate any value greater than 200. In any case, no more than 100 elements have to be checked to satisfy the search condition.

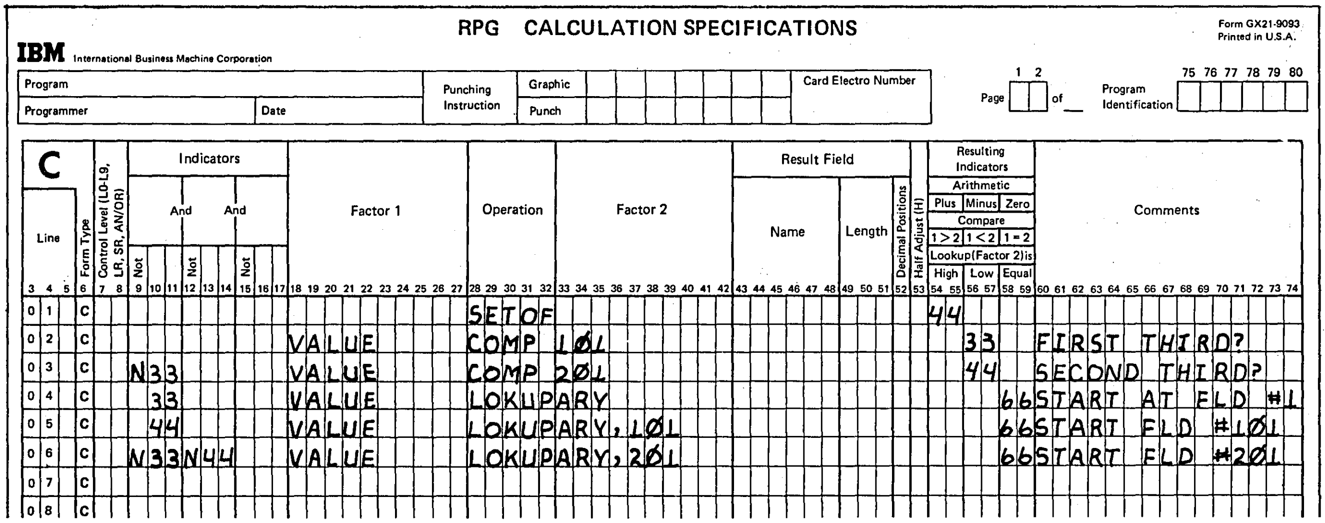
For this example, the number of the array element at which the search is to begin will vary, depending on the value being searched for. Figure 9-33 shows that three LOKUP's have been coded. Only one of the lookup operations is performed for a particular value.

To determine which LOKUP (line 04, 05, or 06) is performed, you must first determine in which part of the array the value is located. The first COMP (compare) operation (line 02) checks for a value in the first 100 elements. If the value is less than 101, indicating the first one third of the array, indicator 33 is set on. If 33 is on, the LOKUP beginning at element 1 is performed (line 04). However, if the value is not in the first third of the array (33 off), another compare (line 03) is necessary to determine if the value is in the second third of the array (indicator 44 set on). Thus, the LOKUP beginning at element 101 (line 05) is performed with indicator 44 on. If neither indicator (33 or 44) was set on, the value must be in the last third of the array, if it is in the array at all. Therefore, with both 33 and 44 off, the LOKUP beginning at element 201 is performed.

For the first LOKUP (line 04), it is not necessary to actually specify the numeric value 1 as the index, in the same manner as 101 is specified for the second LOKUP. When no index is specified with the array name, the search automatically begins at the first field, as if the index were 1.

Note: Setting off indicator 44 (line 01) prevents an error in the lookup function. What would happen if the SETOF operation was not used and indicator 44 was set on in the first cycle and 33 in the second cycle? In that case, 44 would not be set off in the second cycle because the N33 condition would not be satisfied in line 03. Thus, both lines 04 and 05 would be executed. The LOKUP operation in line 04 would be successful and indicator 66 would turn on. The LOKUP operation in line 05 would not be successful and 66 would be turned off. Thus, a not-found condition would result even though the LOKUP was successful.

If the value of the index changes, as in this case, you can use an index field to contain the number of the array field, rather than using the actual number. In this way, it is necessary to code only one LOKUP. Of course, you must place the appropriate number in the index field every time before the lookup operation is performed. Thus an index field will not always reduce the number of specifications required.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	High	Low	Equal	Comments
01	C			SETOF				44			
02	C		VALUE	COMP	1Ø1				33		FIRST THIRD?
03	C	N33	VALUE	COMP	2Ø1				44		SECOND THIRD?
04	C	33	VALUE	LOKUP	ARY					66	START AT FLD #1
05	C	44	VALUE	LOKUP	ARY,1Ø1					66	START FLD #1Ø1
06	C	N33N44	VALUE	LOKUP	ARY,2Ø1					66	START FLD #2Ø1
07	C										
08	C										

Figure 9-33. LOKUP with an Actual Index

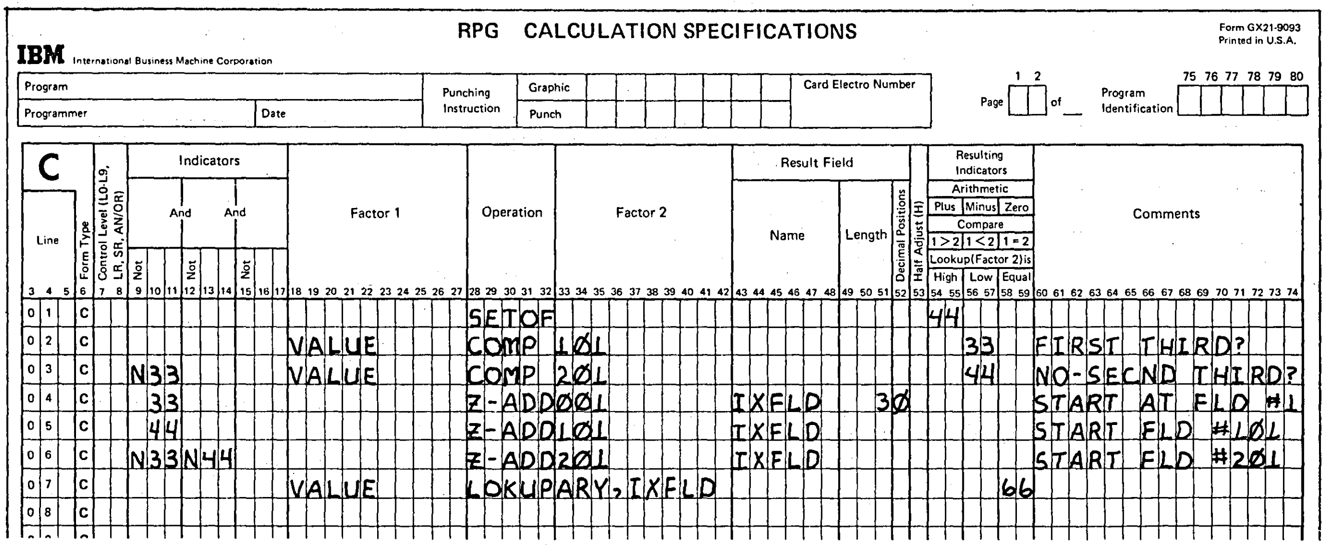
As shown in Figure 9-34, first the compare operations are performed to determine whether the value is in the first, second, or last third of the array. The results of the compare operations determine which number should be zero-added into the index field, IXFLD, before the lookup is performed.

See the previous *Note* for an explanation of the use of the SETOF operation in Figure 9-34 (line 01).

Determining if a Search Is Successful

At this point, we should discuss the index field and how its contents are changed as a result of the lookup operation. Before the lookup is performed, you determine the value which is to be placed in the index field. The array search then begins at the element number specified. The array lookup continues, one element at a time, until the search condition is satisfied or the end of the array has been reached, whichever occurs first. If an index field is specified, the number of the array element first satisfying the search condition is stored in the index field. However, if the end of the array is reached and none of the elements satisfy the search, a *1* is placed in the index field. In any case, if an actual number, not an index field, is specified as the index, the actual index is not changed to reflect the success of the search.

The way in which you determine a successful search is whether the resulting indicator assigned has been turned on or off. Thus, if the resulting indicator is off and an index field had been specified, the index field should contain the value *1*, the result of an unsuccessful search. If the first field of an array satisfied the search condition, the index field would also contain the value *1*; however, in such a case, the resulting indicator would be on.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page of
Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	And	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Plus / 1>2 / High	Minus / 1<2 / Low	Zero / 1=2 / Equal	Comments
01	C						SETOF						44			
02	C					VALUE	COMP	1Ø1						33		FIRST THIRD?
03	C		N33			VALUE	COMP	2Ø1						44		NO-SECND THIRD?
04	C		33				Z-ADD	ØØ1	IXFLD	3	Ø					START AT FLD #1
05	C		44				Z-ADD	1Ø1	IXFLD							START FLD #1Ø1
06	C		N33	N44			Z-ADD	2Ø1	IXFLD							START FLD #2Ø1
07	C					VALUE	LOKUP	ARY,IXFLD							66	
08	C															

Figure 9-34. LOKUP with an Index Field

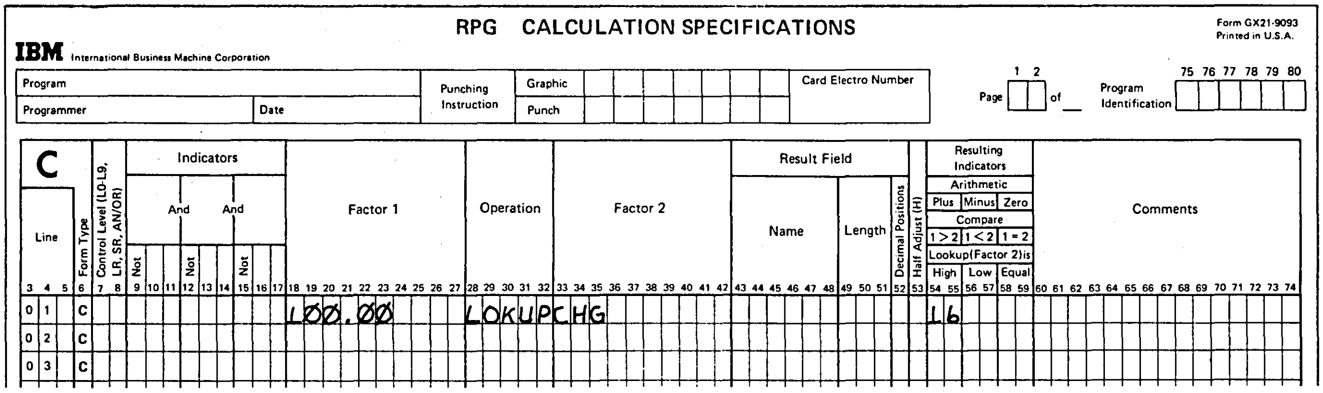
Referencing an Element Which Satisfies a Search

After a successful search, you can use the data from the element which satisfied the condition only if the array name *with an index field* is specified in the LOKUP specifications. If an index field is specified, the number of the field which satisfied the search is stored in the index field. Therefore, specifying the array name with the index field in a subsequent calculation or output specification refers to the element which satisfied the search.

However, if no index field is available (array name specified alone or with a numeric index), the number of the element cannot be determined and, therefore, the data cannot be referenced. You can only determine if one of the array elements does contain the data for which you searched, according to the on-off status of the resulting indicator.

The ability to reference a data item which satisfies a search is one of the major differences between an array lookup and a table lookup. During a table lookup, when a field is found which satisfies the search, the table name alone refers to the data item which satisfied the search. Following a lookup for an array, specifying the array name alone refers to the entire array, rather than to any particular element. The only way an individual array element can be referenced is by specifying the array name with an index.

Assume you wish to search an array CHG to check for amounts over $100. If you only want to determine if there are any elements containing a greater amount, the search can be coded as shown in Figure 9-35. If indicator 16 is on, indicating a successful search, you can then print a message stating there is a charge over $100. Otherwise, if indicator 16 is off, you can print a message stating all charges are under or equal to $100. With the LOKUP specification shown, however, you would have no way of knowing how many elements or which elements satisfied the search condition.



RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

IBM International Business Machine Corporation

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (53)	Resulting Indicators High (54-55)	Low (56-57)	Equal (58-59)	Comments (60-74)
01	C			1ØØ.ØØ	LOKUP	CHG					16			
02	C													
03	C													

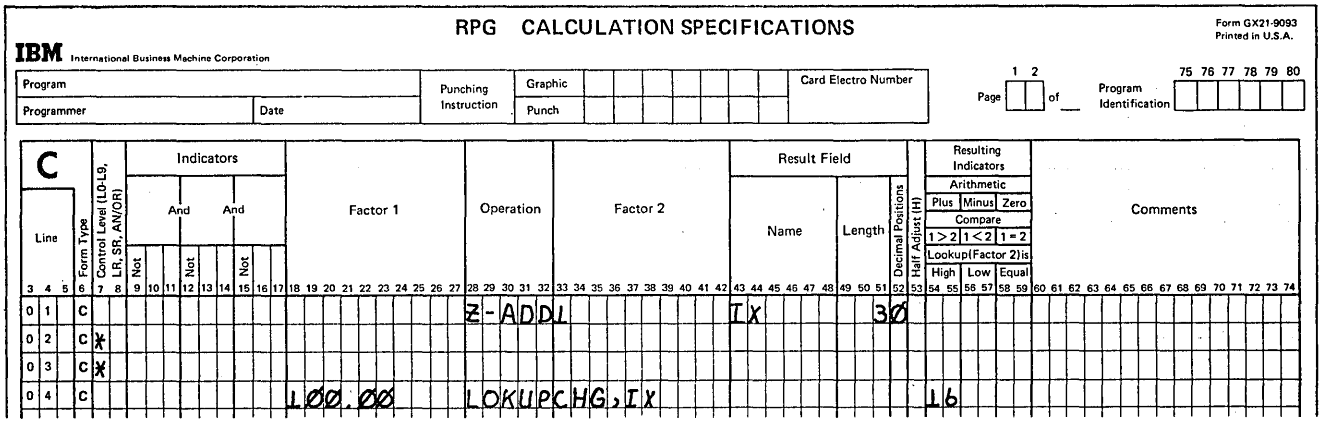
Figure 9-35. Determining Only if a Search Is Successful

If you wish to know which element satisfied the search or, perhaps, how much over $100 the amount is, the array lookup should be coded with an index field (Figure 9-36). The index field can be preset to contain the value *1*, so the search begins at the first element of the array. If the search is satisfied, IX will contain the number of the first element over $100; and the resulting indicator will be turned on. The contents of IX can then be printed to indicate which element satisfied the search. The actual contents of that element can be printed by specifying the array name with the index field (Figure 9-36, Output sheet).

Searching An Array for More Than One Element

The previous example points out an important consideration: an array LOKUP operation is completed when the first element is found which satisfies the search condition. If you wish to find all elements which satisfy the condition, you must code additional specifications which cause the program to loop back in calculations to repeat the lookup operation from the point where the last search was successful.

As an example, assume your company manufactures 25 different items, identified by item codes 1-25. A 25-element array QTY (Figure 9-37) is used to keep track of the quantity in stock of each item. The first element contains the quantity of item code *1*, the second element contains the quantity of item code *2*, and so on.

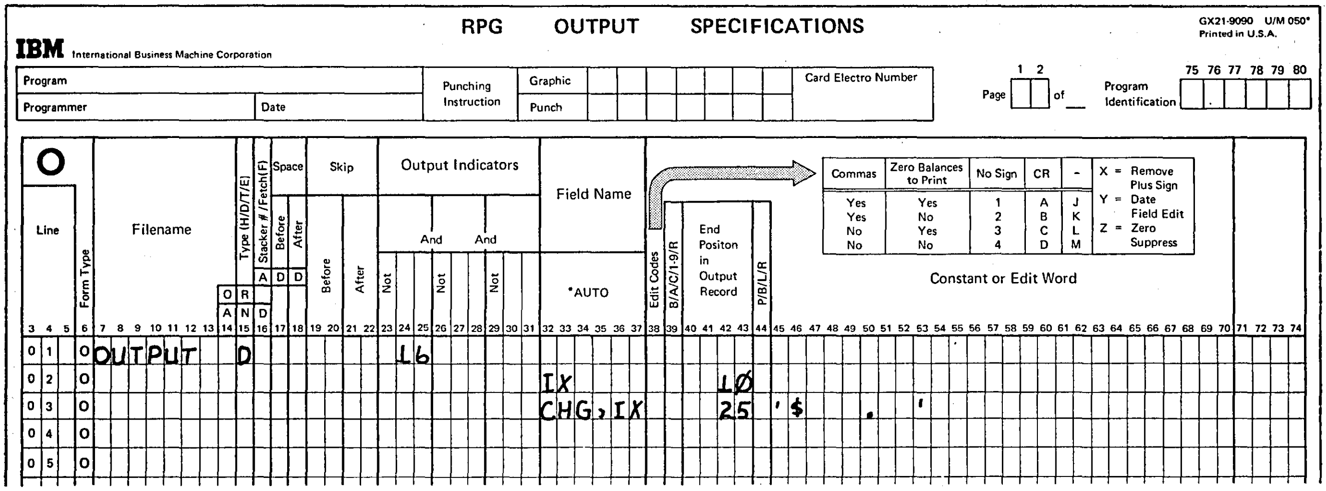


IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Resulting Indicators	Comments
01	C				Z-ADD	1	IX	3	Ø		
02	C	*									
03	C	*									
04	C			1ØØ.ØØ	LOKUP	CHG,IX				16	



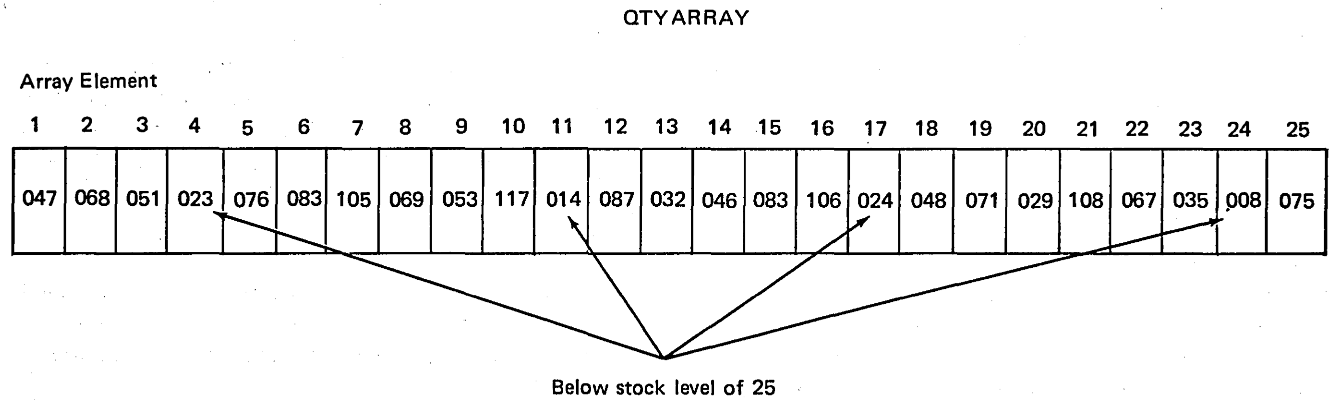
IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Space	Skip	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	OUTPUT	D			16				
02	O						IX		1Ø	
03	O						CHG,IX		25	'$. '
04	O									
05	O									

Figure 9-36. Determining Which Array Element Satisfies the Search



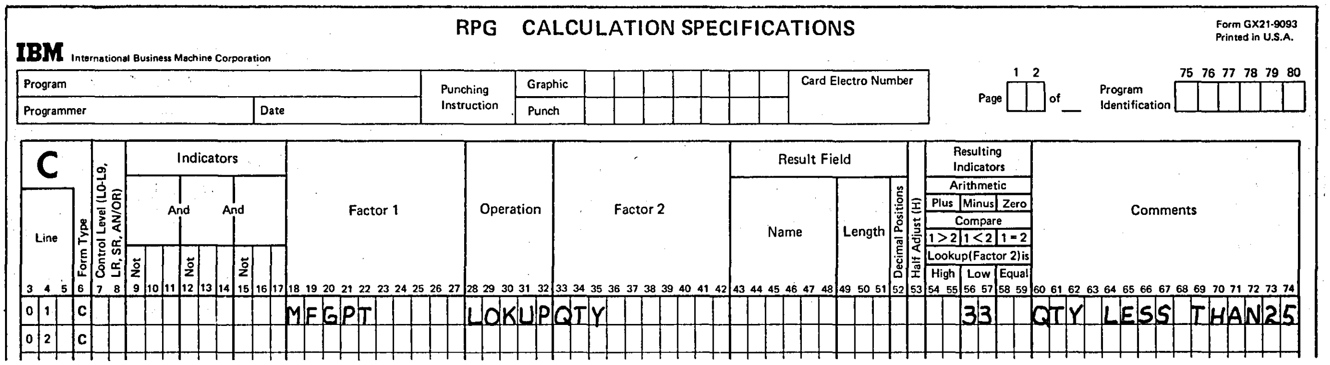
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
047	068	051	023	076	083	105	069	053	117	014	087	032	046	083	106	024	048	071	029	108	067	035	008	075

Figure 9-37. More than One Array Element Which Satisfies the Search Condition

100 items are to be manufactured and added to stock whenever the quantity of an item falls below 25. To determine which items are to be manufactured, every week the QTY array is searched, comparing the array elements with a search word (MFGPT) of 25 from a data card. When a quantity is found to be less than 25 (search condition Low), the item code and quantity in stock are printed.

From Figure 9-37 you can see that four items must be manufactured. The specifications in Figure 9-38 will not locate all of the items with quantities less than 25. The lookup operation shown will locate only the first quantity below 25. If only one data card (containing the search word) is read, the specifications are performed once. As you know, for every data card read, the program cycle is repeated. However, even if several data cards with the same search word are read, every time the lookup is repeated, the search begins again at element *1* of the array. Therefore, the same array element satisfies the search every time, and the other three quantities are never found.

To locate more than one element satisfying the same search condition, the LOKUP must be repeated within a single program cycle. Not only must the LOKUP be repeated, but the search must begin at the point where the previous search ended. You can repeat the LOKUP using the GOTO and TAG operations, as shown in Figure 9-39. To make sure the repeated search begins where the last search left off, you must specify the array name *with an index field* in the LOKUP specification. The contents of the index field is then updated after each successful search to indicate at which array element the next search should begin.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page □□ of ___ Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High (1>2)	Low (1<2)	Equal (1=2)	Comments
01	C			MFGPT	LOKUP	QTY						33		QTY LESS THAN25
02	C													

Figure 9-38. Array Search Which Locates Only One Element

The first search should begin at element *1*. Thus, as Figure 9-39 shows, the index field IX is initially set up to contain the value *1*. (The field is zeroed before adding *1*, since you have no way of knowing the contents of IX at the beginning of the program run.) The TAG operation is not performed; therefore, the computer skips this specification and performs the LOKUP.

When the first QTY element less than 25 is found, the number of the element (04) is placed in the index field. Providing the LOKUP was successful (33 on), a *1* is added to the value in the index field, to indicate at which element the next search should begin. The value in the index field is then compared to 26 to see if the entire array (25 fields) has been searched. If there are still array elements to be checked (indicator 44 on), the program branches back (GOTO) to perform the LOKUP again. The search would then begin again, only at the element following the last element which satisfied the search. The calculation specifications would be repeated over and over until all items to be manufactured are located and until the end of the array is reached.

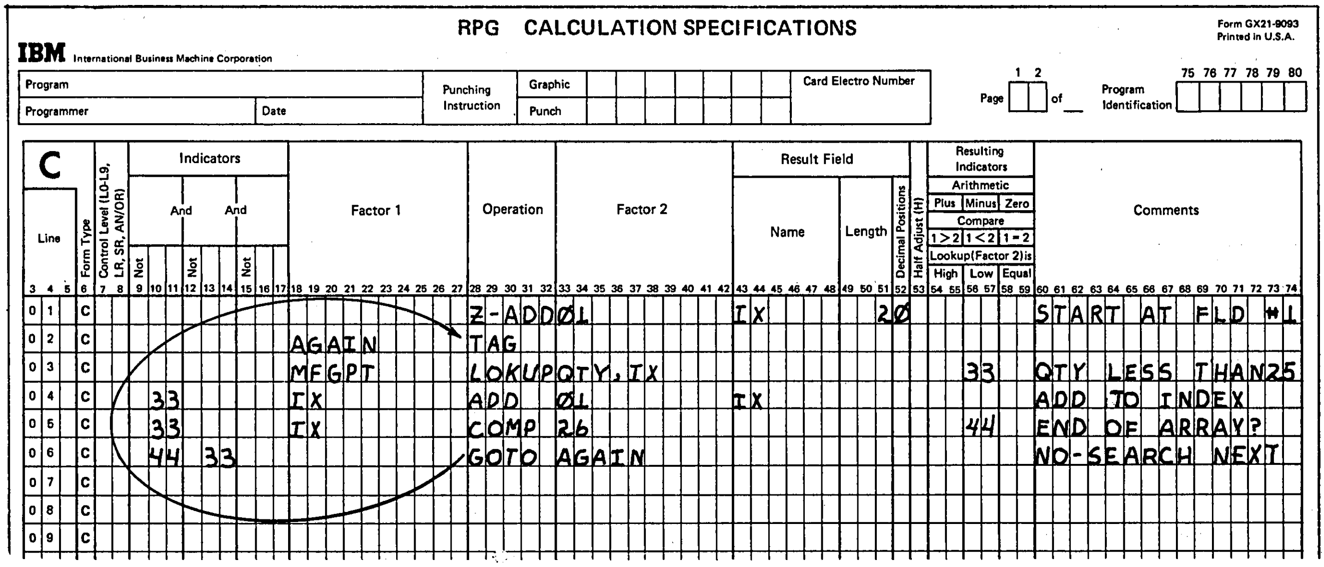
Output During an Array Search

The specifications in Figure 9-40 search through the QTY array to locate more than one element. In this case, it does no good to search through an array unless you know what data was found. For this reason, each quantity less than 25 and its related item code are printed. Following each successful search, the item code number (same as the number of the array element containing the quantity) is stored in the index field IX. Thus, the field IX can be printed. The actual quantity which satisfied the search can be printed by specifying the array name with the index field in the output specification.

Since the output specifications usually are not performed until all calculations are done, normal output would be invalid since there would be an attempt to reference array element 26.

In order to print each item code (field number) located in the array search (and its quantity), output must be done before the contents of the index field are changed.

You have learned that using the EXCPT operation on the Calculation sheet makes it possible to perform output specifications before calculations are finished and then to return to finish the calculation operations. As Figure 9-40 shows, following a successful LOKUP, the EXCPT operation then causes the data placed in the index field to be printed, followed by the contents of the array element which satisfied the search. After the exception output has been performed (output lines identified by an E in column 15), the program continues with the calculation specifications by executing the calculation which follows the EXCPT operation.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number
Programmer | Date
Page 1 2 of ___ | Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level	Indicators 9-11	Indicators 12-14	Indicators 15-17	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust	High	Low	Equal	Comments
01	C						Z-ADD	Ø1	IX	2	Ø					START AT FLD #1
02	C					AGAIN	TAG									
03	C					MFGPT	LOKUP	QTY,IX						33		QTY LESS THAN25
04	C		33			IX	ADD	Ø1	IX							ADD TO INDEX
05	C		33			IX	COMP	26						44		END OF ARRAY?
06	C		44	33			GOTO	AGAIN								NO-SEARCH NEXT
07	C															
08	C															
09	C															

Figure 9-39. Repeating a LOKUP to Locate All Array Elements Satisfying the Search Condition

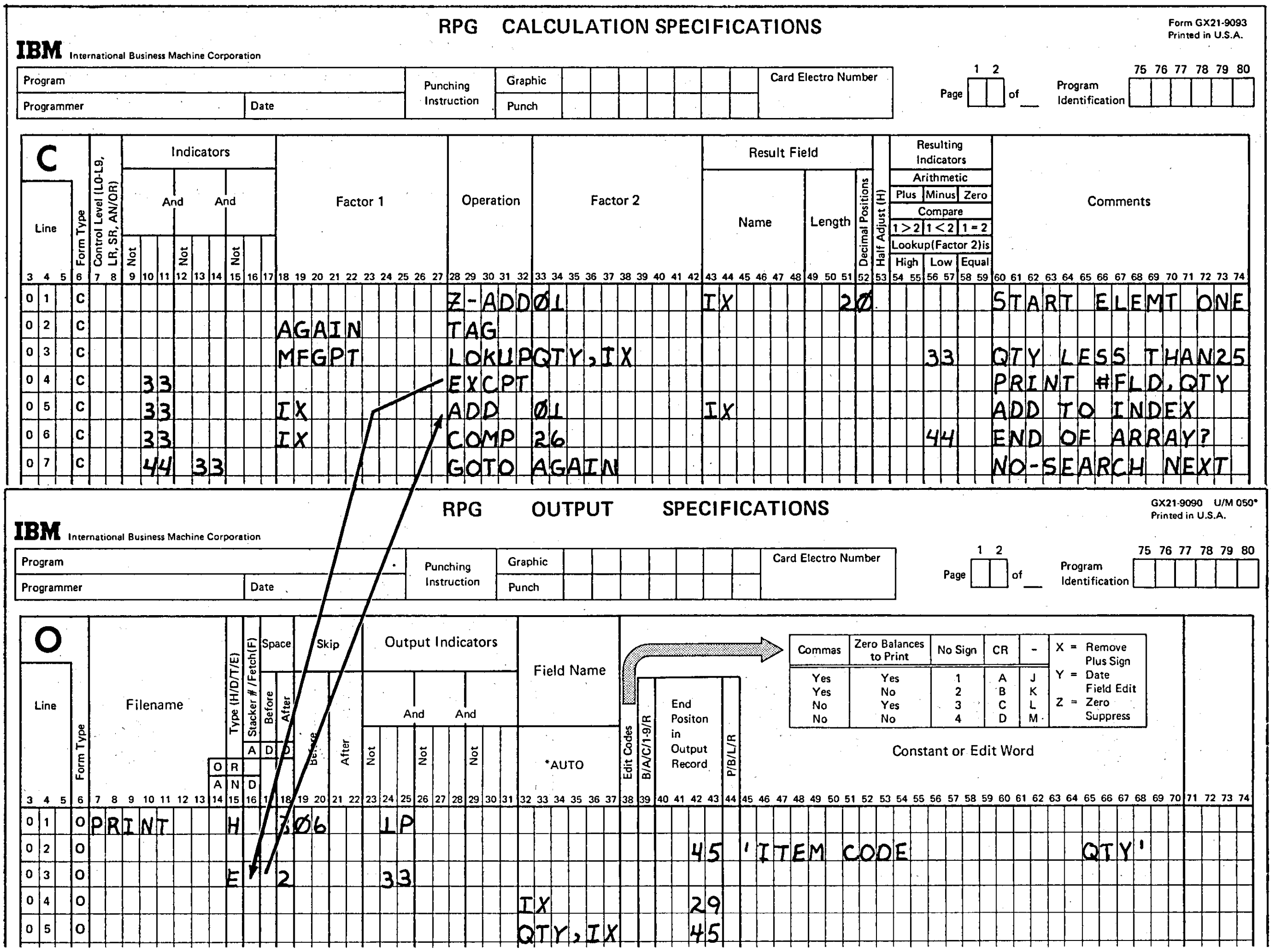
LOADING ARRAYS

In the beginning of this chapter (*Defining an Array*), you learned that arrays are divided into three types based on when the array data is stored into the array. The three different times are compilation time, pre-execution time, and execution time. Data can be stored in any of the following ways:

- *Compilation time:* keyed in on a console (keyboard) device; read from punched cards, tape, or disk source library
- *Pre-execution time:* keyed in on a console device; read from punched cards, tape, or disk
- *Execution time:* Extracted from an input file on a console device, punched cards, tape, or disk during execution of the program
- *Execution time:* Created by calculations performed during the program

Compile Time Arrays

Arrays loaded at the same time as your RPG II source program are referred to as compile time arrays. The array is compiled along with the RPG II source program. The array data actually becomes part of the object program. One definite advantage in creating compile time arrays is that you need not load separate array files into the computer every time you wish to run that object program.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Indicators	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Resulting Indicators High	Low	Equal	Comments
01	C				Z-ADD	Ø1	IX	2	Ø				START ELEMT ONE
02	C			AGAIN	TAG								
03	C			MFGPT	LOKUP	QTY,IX					33		QTY LESS THAN25
04	C	33			EXCPT								PRINT #FLD,QTY
05	C	33		IX	ADD	Ø1	IX						ADD TO INDEX
06	C	33		IX	COMP	26					44		END OF ARRAY?
07	C	44	33		GOTO	AGAIN							NO-SEARCH NEXT

RPG OUTPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type	Space After	Skip Before	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	H	3	Ø6	1P			
02	O							45	'ITEM CODE QTY'
03	O		E	2		33			
04	O						IX	29	
05	O						QTY,IX	45	

Figure 9-40. Output of Array Element as it is Located in the Search

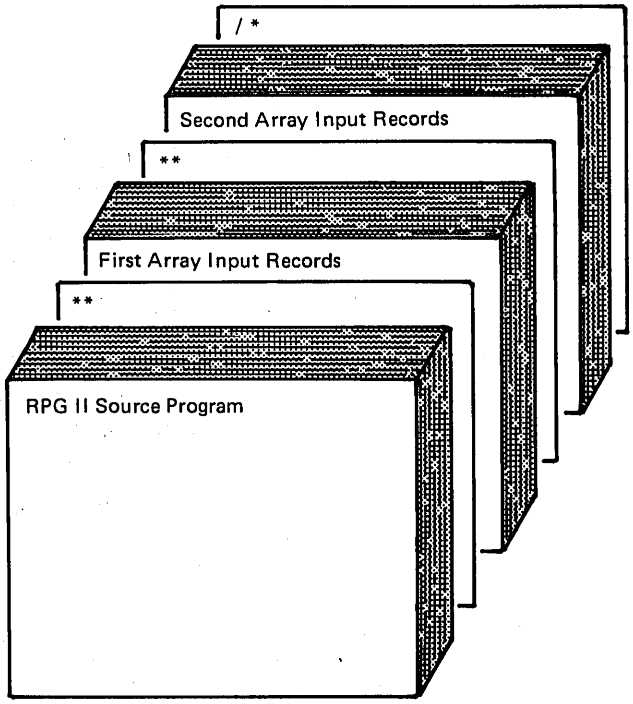
Changing Compile Time Arrays

Temporary changes to data in a compile time array exist only for a particular run and are made as easily as for any array. Calculation specifications which have been previously coded in the program can modify any of the array elements.

Making permanent changes to a compile time array requires recompiling the entire RPG II source program along with the new or changed array input records. The object program produced then contains the current array data.

Loading Compile Time Arrays

An array to be compiled with your program should follow the RPG II source program (Figure 9-41). There should be a record immediately before the array containing ** in positions 1-2. Position 3 must be blank but remaining positions may be used for comments (such as the array name). If more than one array is to be compiled, a ** record should be placed before each array. Furthermore, the compile time arrays must be loaded in the same order as they are described on the Extension sheet. The end of file record (/* in positions 1-2) which usually comes at the end of the source program is then placed after the last compile time array.



Model 10 Card System Users: The source program and array input records as shown here are placed in the secondary MFCU hopper. The RPG II Compiler program is placed in the primary hopper.

Figure 9-41. Arrangement of Input for Compile Time Arrays

Model 10 Disk System, Model 15, and Model 6 users may place compile time arrays in the source library following the source program. The same record sequence as shown in Figure 9-41 is used. See the applicable reference manuals foı your system for specific procedures.

Pre-execution Time Arrays

In general, if an array is to be permanently modified often, a pre-execution time array is easier to use than a compile time array. A pre-execution time array is not compiled with your RPG II source program. Instead, once the object program has been loaded into the computer to be executed, the array is loaded separately like an input data file. The array is then used by the object program, rather than being a part of the program.

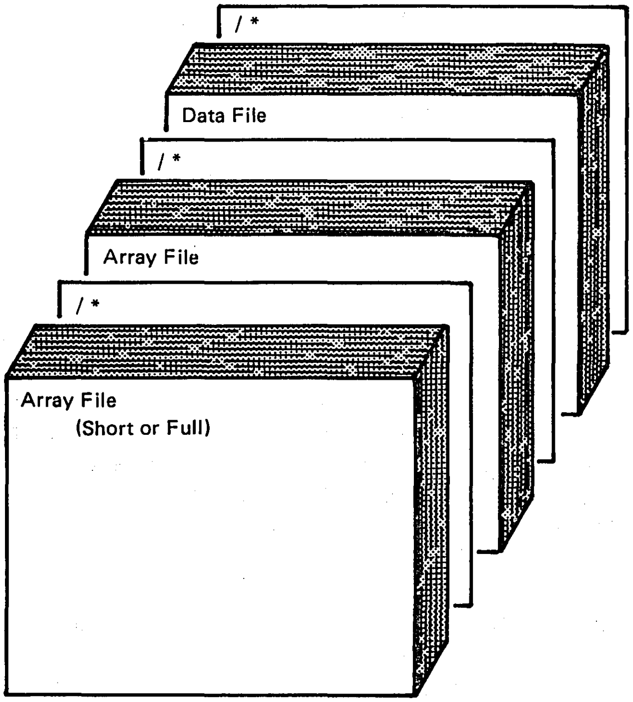
Changing a Pre-execution Time Array

Modifying a pre-execution time array is easier than changing a compile time array. Modifying the contents of the array permanently (whether a short array or a full array) can be done by inserting and deleting change records. In any event, only the array file is changed; there is no need to make changes in the RPG II object program.

Loading Pre-execution Time Arrays

Pre-execution time arrays are similar to any other input data files in that the RPG II object program uses the files when the program is executed. Unlike other data files, however, pre-execution time arrays are read completely before execution of the program continues.

The array files should be in the same order as for compile time arrays. All array files are to be loaded in the same order as they are described on the Extension sheet. Furthermore, if both pre-execution time array files and other input data files are to be used by a program, all arrays must be loaded before the data files. An end of file card (/*) must follow every pre-execution time array file, regardless of whether the array is short or full (Figure 9-42).



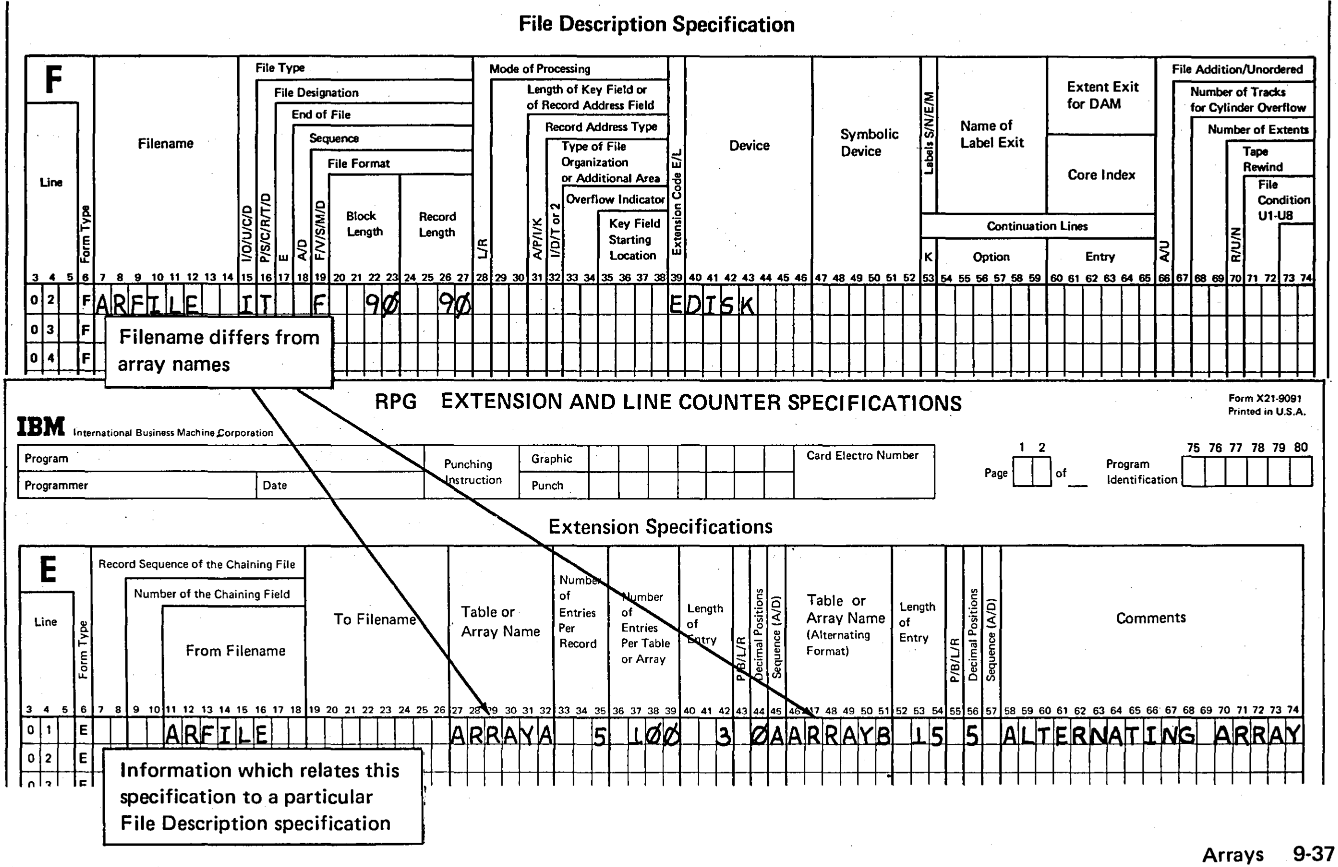
Model 10 Card System Users: The array files are loaded from the secondary MFCU hopper. The RPG II object program is loaded from the primary hopper.

Other Systems: Array files loaded at pre-execution time may be loaded from console, cards, disk or tape.

Figure 9-42. Arrangement of Input for Pre-execution Time Arrays

Specifications for Pre-execution Time Arrays

Since a pre-execution time array is a separate file to be used by the program, the entire file of array input records must be defined on the File Description sheet, just as any other file must be. Figure 9-43 shows the file description specifications required to define a pre-execution time array input file. A filename, different from the array name, should be assigned to the entire array file (columns 7-14). A unique filename should be assigned because the single file may actually contain data for two arrays, if alternating format records are used. Figure 9-43 shows a file called ARFILE, containing data for both ARRAYA and ARRAYB. An I in column 15 says that ARFILE is an input file. Notice also, that the File Designation entry (column 16) must be a T, to indicate that this is an array file, as well (T stands for either a table file or an array file). The Device entry (columns 40-46) indicates the device from which the array file is read (for the Model 10 Card System, the entry must be MFCU2).



File Description Specification

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	Mode of Processing	Extension Code E/L	Device
02	F	ARFILE	I	T			F	90	90		E	DISK
03	F											
04	F											

Filename differs from array names

IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Extension Specifications

Line	Form Type	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	Decimal Positions	Comments
01	E	ARFILE		ARRAYA	5	1ØØ	3	Ø	A	ARRAYB	15	5	ALTERNATING ARRAY
02	E												

Information which relates this specification to a particular File Description specification

Figure 9-43. Defining a Pre-execution Time Array File

Ordinarily, if an input file is to be in a particular sequence, an entry (A or D) is made in column 18 of the File Description sheet. When specifying a sequence for array files, however, the sequence columns (45 and 57) on the Extension sheet must be used, rather than the sequence column on the File Description sheet. An E has been entered in column 39 of the File Description sheet to indicate that the records in this array file are further described on the Extension sheet.

Looking at Figure 9-43, you can see that the filename assigned on the File Description sheet is also entered under From Filename (columns 11-18) of the Extension sheet. This common entry tells the computer that the extension specifications describing ARRAYA and ARRAYB arrays are associated with the ARFILE file defined on the File Description sheet. For compile time arrays, previously described, no entry is made in columns 11-18, since no filename is assigned to compile time array records.

Storing Input Data into Execution Time Arrays

When an execution time array is defined in an extension specification, an array is set up in the computer, ready to receive array data. You have learned that the array data can either be generated by calculations in your program or be taken from an input file that your program reads. There is an important difference between the input file used to build an execution time array and the input file used to build a pre-execution time array (see *Loading Arrays, Pre-execution Time Arrays*). That is, the input file from which an execution time array is built is not a special array file designated by a T in column 16 of the File Description sheet. Therefore, the array data is not automatically loaded into the array at the beginning of execution time. Instead, you must describe the input data to be loaded into the array on input specifications and ensure that the necessary data is in the array before doing operations that use the array data.

Fields of array data to be read from input records must be described on the Input sheet. The input specifications indicate where the data is located on the record. How the array information is described and stored depends on three factors:

1. How the array data is organized on a record.

2. Whether the data for an array is contained in one or more records.

3. Which System/3 model you are using.

An input record containing array data can contain only data for that array or can contain both array data and other data fields to be used in the program. In either case, the array data is organized in one of two ways:

1. All array elements may occupy consecutive positions on the record; that is, each element immediately following another with no blanks or other data between the elements.

2. The array elements may be scattered on the record, in any order, with blanks or normal input fields placed between the array elements.

The way in which the data is organized and the size of the array generally determines the number of input records required to contain the array data.

Array Data in Consecutive Positions on One Record

If array elements are in order in consecutive positions on a record, describing and storing the data is very easy. All of the array data on the one record may be described on the Input sheet as if it were a single field. Thus, only one input specification is necessary to indicate a name for the field and the columns on the record where the array data begins and ends (Figure 9-44). By specifying the name of the array as the field name, the data is automatically stored in the appropriate elements of the array as the input record is read.

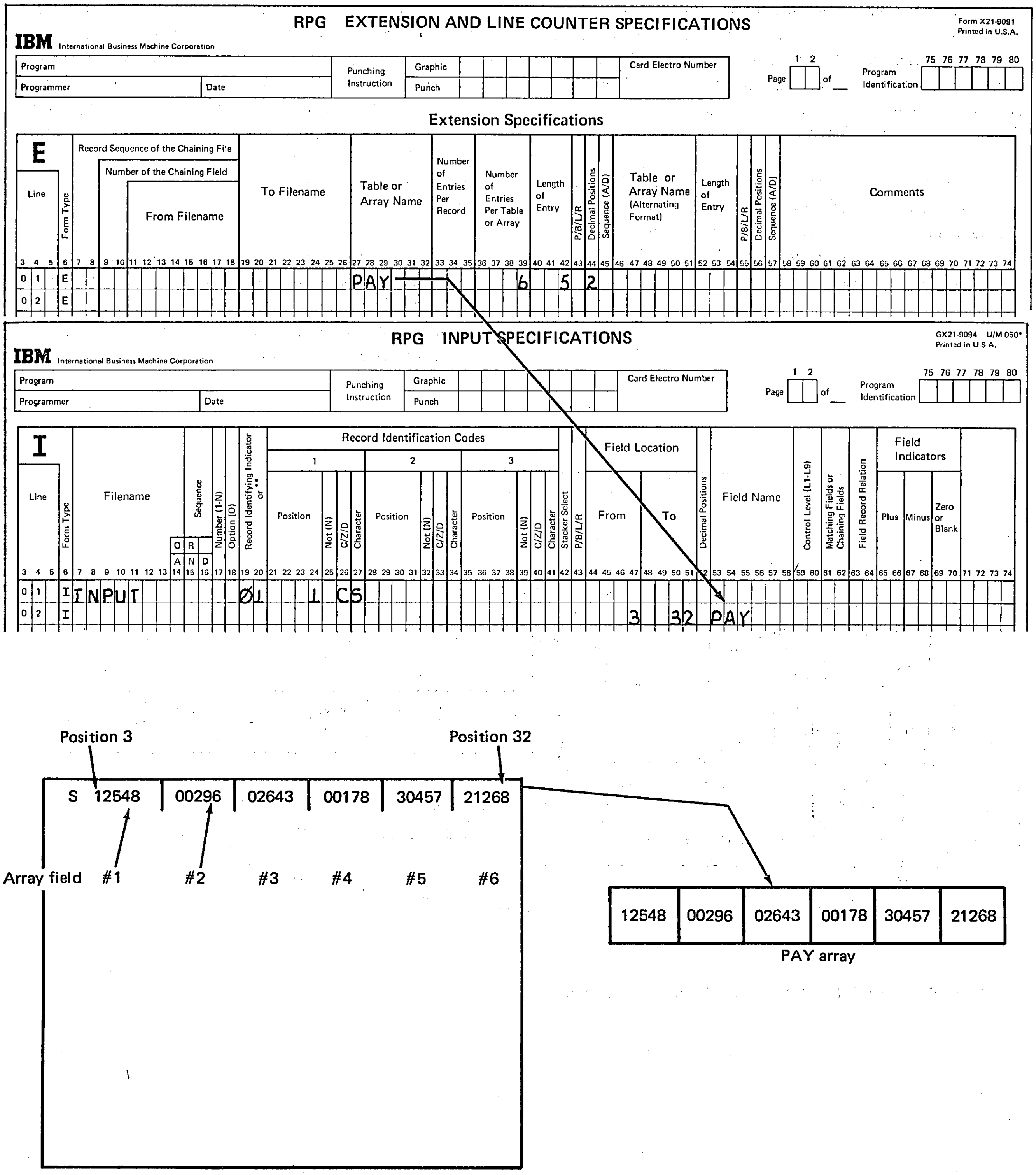
When you describe an input record of array data, you specify no entry in column 52 (Decimal Positions) of the Input sheet. Since the array name and characteristics have been previously defined on the Extension sheet, the entry in column 44 of the Extension sheet indicates the number of decimal positions in each array element.

Array Data Scattered on One Record

When array elements are scattered on an input record, each field must be described separately on the Input sheet to indicate where each item of array data begins and ends. Two methods are available to load the data into the array:

1. Assign a unique field name to each field of array data on the input record, then code calculations to move each data field individually into the appropriate array element.

2. Assign the array name with the proper index to each field of array data *in the input record* and the array will be loaded automatically as the data is read. (This method of loading array data is not available for the System/3 Model 10 Card System.)



IBM International Business Machines Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091 Printed in U.S.A.

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			PAY		6	5		2							
02	E															

IBM International Business Machines Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Position	Not (N)	C/Z/D	Character	Field Location From	Field Location To	Decimal Positions	Field Name
01	I	INPUT				Ø1	1		C	S				
02	I										3	32		PAY

Figure 9-44. Storing Data in an Array

Assume that a 6-field array named EMP is set up by coding extension specifications. The six fields of data for the array are scattered on a record, as shown in Figure 9-45. Additional input information (blanks and other input fields) is recorded between the array fields. Furthermore, the array fields are not in the order in which they are to be stored.

When you describe the array data, you must identify each field by a separate line of input specifications, because the array data is not continuous. Normal input fields can be described along with the array fields. Separate fields can be identified and stored as follows:

All System/3 Models: One way to identify and store array data is shown in Figure 9-46. Unique field names are assigned to individual fields of array data on the Input sheet. Once the scattered fields have been described on the Input sheet, each field of array data is stored in the array using a MOVE calculation. Since each field has a unique field name and must be stored in a specific array element, a separate move specification must be coded for each field to be stored.

The specifications which move the array data into the array elements should generally be specified first on the Calculation sheet. This ensures that the data will be in the array when any calculations on the array (specified later on the Calculation sheet) are performed.

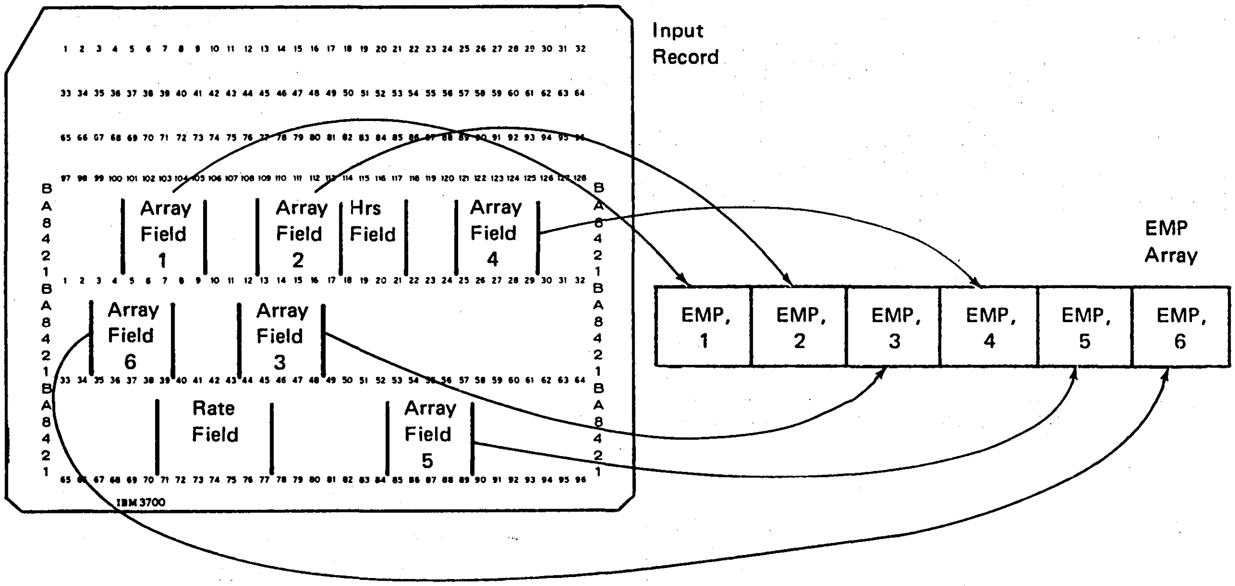
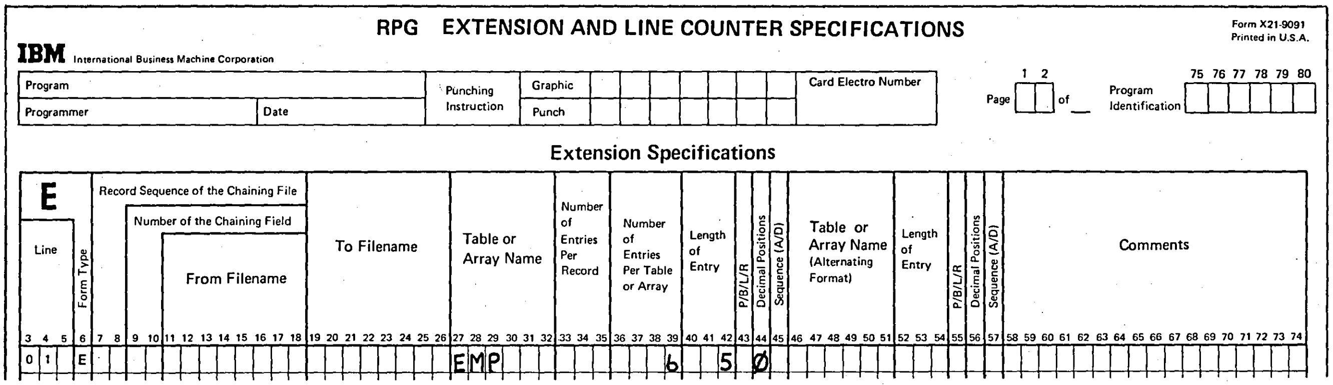


Figure 9-45. Scattered Fields of Array Data in One Input Record



IBM International Business Machine Corporation

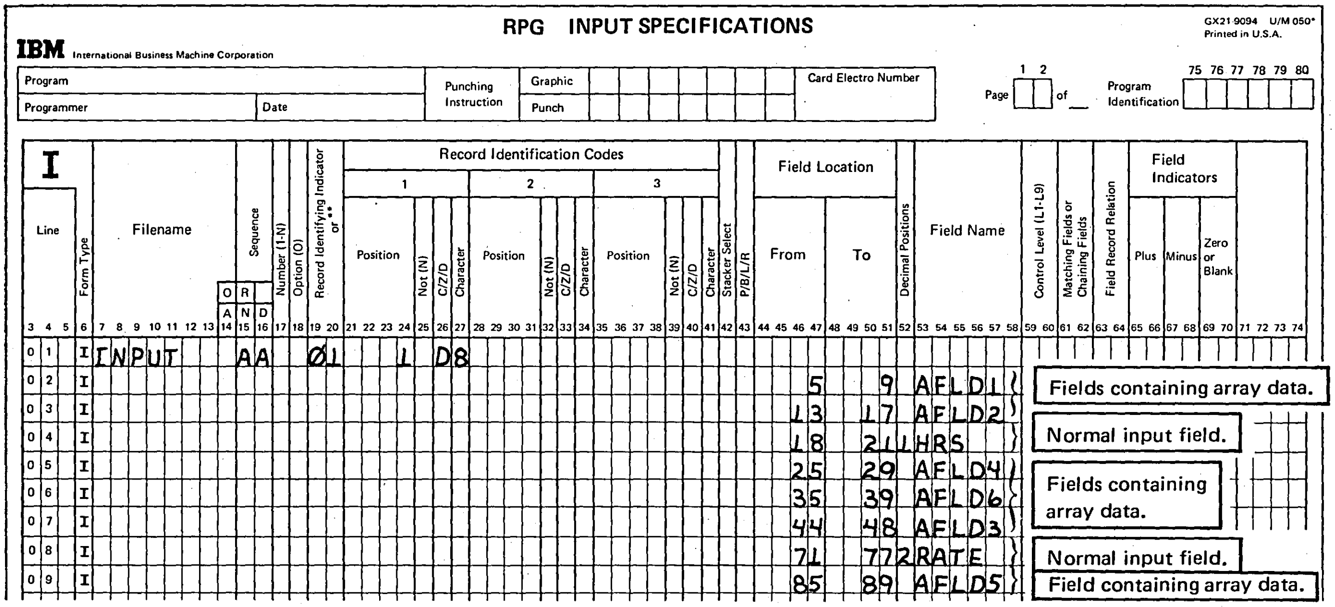
RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification
Programmer | Date

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			EMP		6	5		Ø							



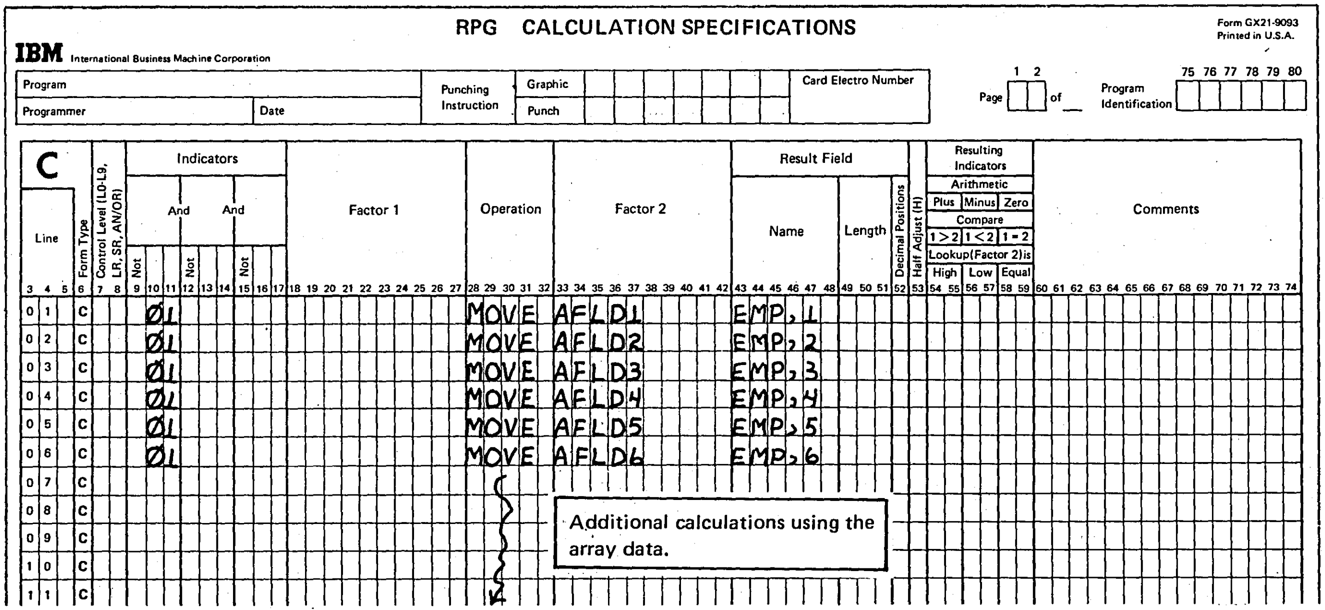
IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification
Programmer | Date

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Record Identification Codes 1: Position	Not (N)	C/Z/D	Character	Field Location From	To	Decimal Positions	Field Name	
01	I	INPUT	AA			Ø1	1		D	8					
02	I										5	9		AFLD1	Fields containing array data.
03	I										13	17		AFLD2	
04	I										18	21	1	HRS	Normal input field.
05	I										25	29		AFLD4	Fields containing array data.
06	I										35	39		AFLD6	
07	I										44	48		AFLD3	
08	I										71	77	2	RATE	Normal input field.
09	I										85	89		AFLD5	Field containing array data.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification
Programmer | Date

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C	Ø1		MOVE	AFLD1	EMP,1					
02	C	Ø1		MOVE	AFLD2	EMP,2					
03	C	Ø1		MOVE	AFLD3	EMP,3					
04	C	Ø1		MOVE	AFLD4	EMP,4					
05	C	Ø1		MOVE	AFLD5	EMP,5					
06	C	Ø1		MOVE	AFLD6	EMP,6					
07	C										
08	C										
09	C										
10	C										
11	C										

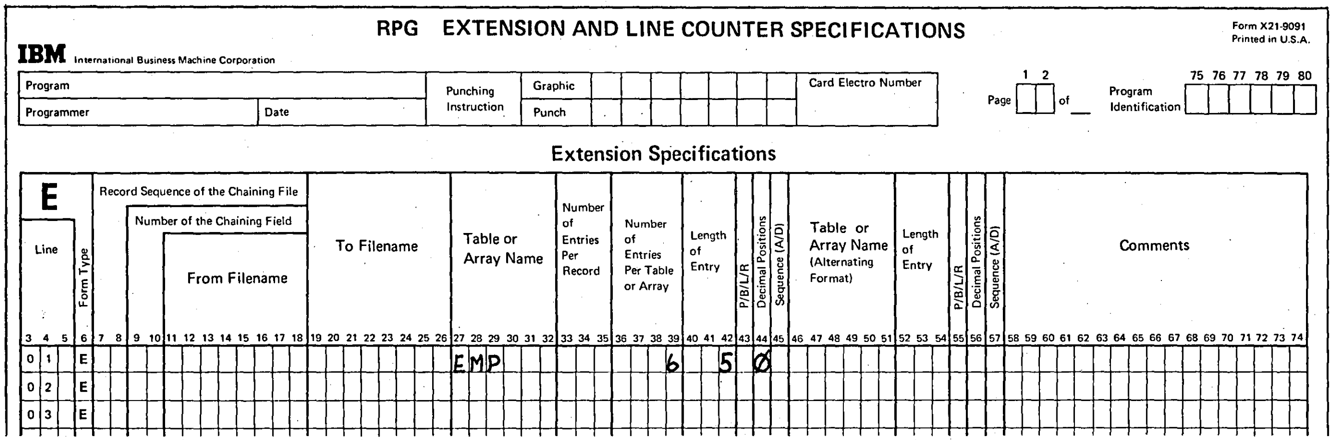
Additional calculations using the array data.

Figure 9-46. Loading Array Data From Scattered Fields by Assigning Unique Field Names

All System/3 Models (Except Model 10 Card System and Model 15C): Figure 9-47 shows a second way to load an array from scattered fields. The array name with an index is assigned to each field of data in the input record. In this way, the data is loaded directly into the array as the fields are read. No move calculations are necessary.

Array Data Consecutive on More Than One Record (Model 10 Card System)

Consider a case where the array data on all input records is organized consecutively. Data for a 25-element array named TAX is contained on two input records (Figure 9-48). The first record contains 19 fields, the second record, six. Each numeric field is five characters long. The data is organized on the records in the order it is to be stored in the array.



IBM International Business Machine Corporation

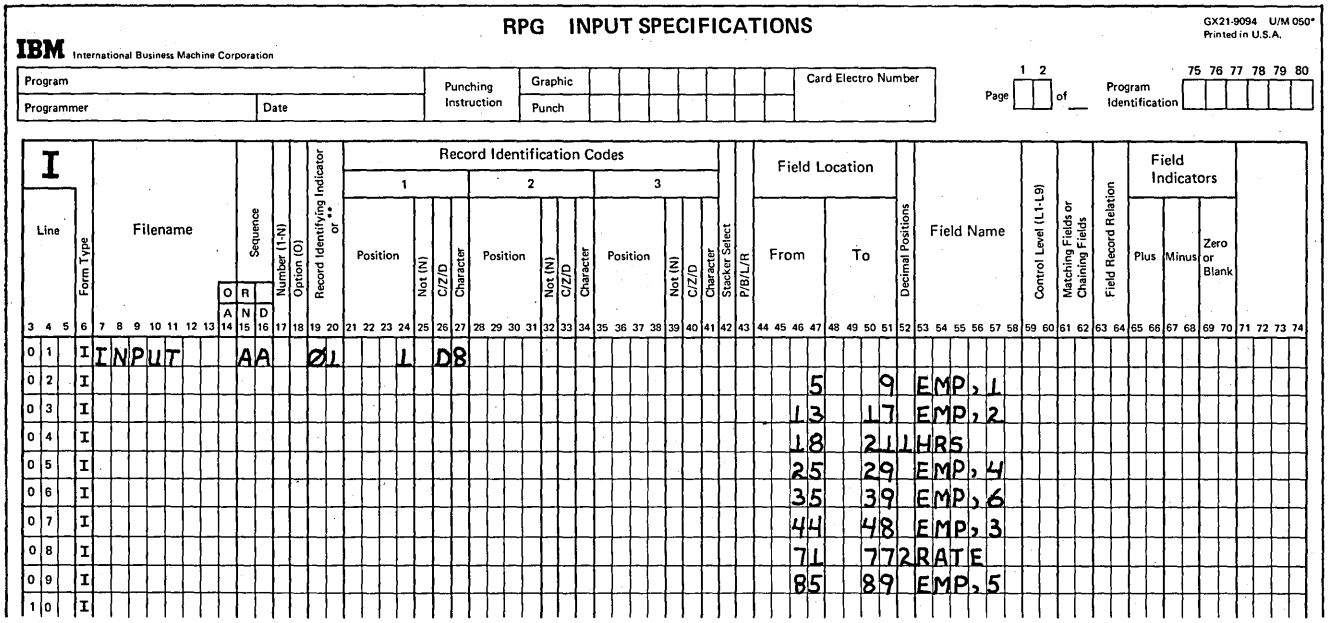
RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E					EMP		6	5		Ø							
02	E																	
03	E																	



IBM International Business Machine Corporation

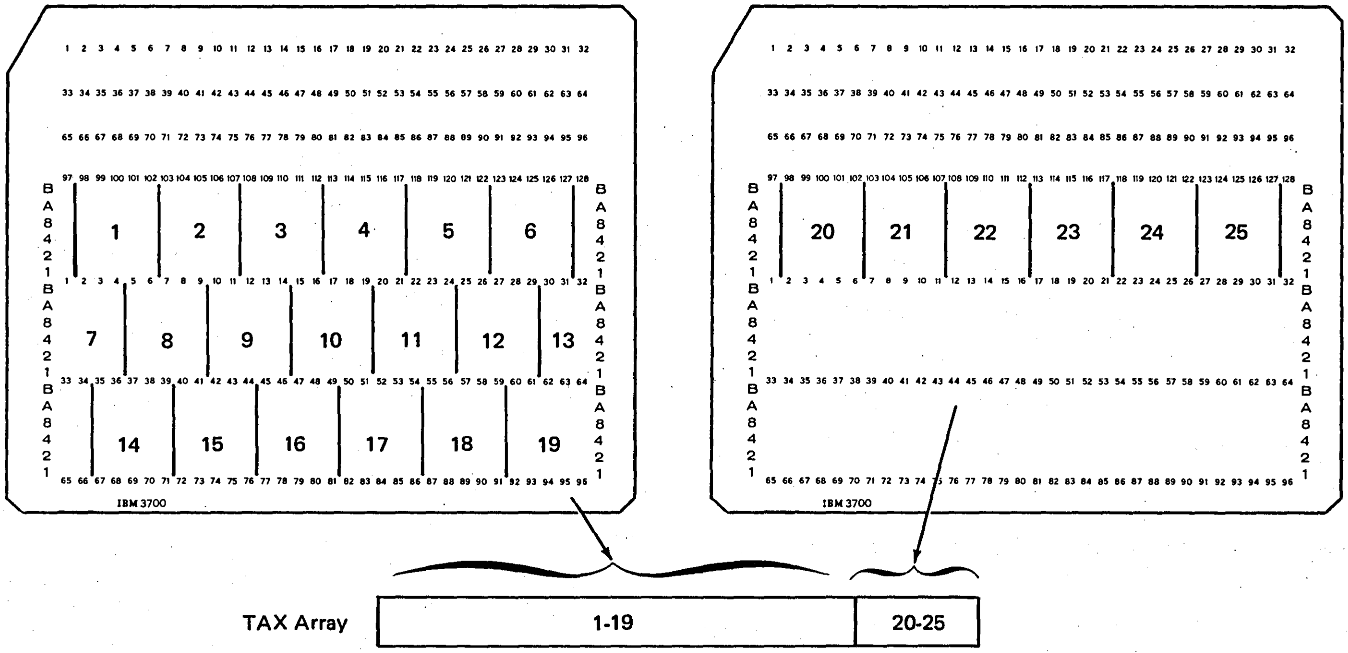
RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position (1)	Not (N)	C/Z/D	Character	Position (2)	Not (N)	C/Z/D	Character	Position (3)	Not (N)	C/Z/D	Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank
01	I	INPUT	AA			Ø1	1		D	8																				
02	I																				5	9		EMP,1						
03	I																				13	17		EMP,2						
04	I																				18	21	1	HRS						
05	I																				25	29		EMP,4						
06	I																				35	39		EMP,6						
07	I																				44	48		EMP,3						
08	I																				71	77	2	RATE						
09	I																				85	89		EMP,5						
10	I																													

Figure 9-47. Loading Array Data From Scattered Fields by Means of Input Specifications



RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

IBM International Business Machine Corporation

Form X21-9091
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page 1 2 of __
Program Identification 75 76 77 78 79 80

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					TAX		25	5									

RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program
Programmer Date
Punching Instruction Graphic Punch
Card Electro Number
Page 1 2 of __
Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-13)	Sequence (14-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	1 Position (21-24)	1 Not (N) (25)	1 C/Z/D (26)	1 Character (27)	2 Position (28-31)	2 Not (N) (32)	2 C/Z/D (33)	2 Character (34)	3 Position (35-38)	3 Not (N) (39)	3 C/Z/D (40)	3 Character (41)	Stacker Select (42)	P/B/L/R (43)	Field Location From (44-47)	Field Location To (48-51)		Zero or Blank (69-70)	(71-74)
01	I	INPUT	AA			Ø1	1		D	1															
02	I																				2	96	TAX		
03	I		AA			Ø2	1		D	2															
04	I																				2	6	TAX,2Ø		
05	I																				7	11	TAX,21		
06	I																				12	16	TAX,22		
07	I																				17	21	TAX,23		
08	I																				22	26	TAX,24		
09	I																				27	31	TAX,25		

The array elements on the second record need not be loaded separately on the Model 10 Disk System, Model 6, or Model 15, since the MOVEA operation code is available on these systems.

Figure 9-48. Loading Array Data Consecutive on More than One Record (Model 10 Card System)

It is important to note that when data is stored in an array by specifying the array name as the field name, the information is placed at the beginning of the array. Thus, the 95 columns of data from this first input record are stored in elements 1-19 of the array (Figure 9-48).

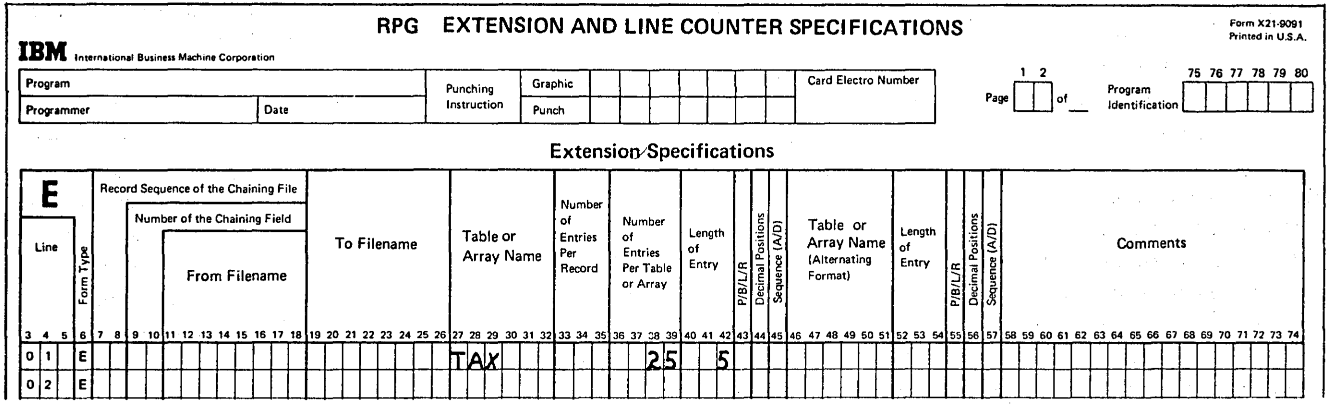
Although the data on the second record is also arranged consecutively, each element is loaded separately. The second record cannot be defined as a single array field and stored automatically in the array because the data would be stored at the beginning of the array, destroying the data previously stored at the beginning of the array. Instead, the data from the second record is loaded by defining the individual fields as array elements on the Input sheet (Figure 9-48). The data could also be loaded by assigning a unique fieldname to each field of array data on the second record and using MOVE operations to move each field to its proper array element. In this case, specifications would be similar to those for the EMP array in Figure 9-46.

In this example, the method of defining and storing data in the TAX array is relatively simple. However, if there are a large number of data fields contained on records other than the first, storing the data can require a great number of coding lines. Suppose, for example, the TAX array consists of 50 fields. Three records are required to contain the data. The first two records contain 19 fields each; the third contains 12 fields. Storing the data using the method shown in Figure 9-48 requires 31 separate lines of coding to load the data on records two and three. Because of this, you might want to consider loading the array as a compile time or pre-execution time array, instead.

Array Data Consecutive on More Than One Record (Model 6, Model 10, Model 12 and Model 15)

On the Model 6, Model 10, Model 12 and Model 15, it is much easier to load array data that is consecutive on more than one input record. Consider the previous example (Figure 9-48). The array data on the second record can be described as a single field on these systems, because the MOVEA operation code is available on these systems to move data from a field to an array. Figure 9-49 shows the coding necessary to load the TAX array when the MOVEA operation is used in calculations.

Using the MOVEA operation, data that is consecutive on many input records can conveniently be loaded during program execution. See your RPG II Reference Manual for a complete description of the use of MOVEA.



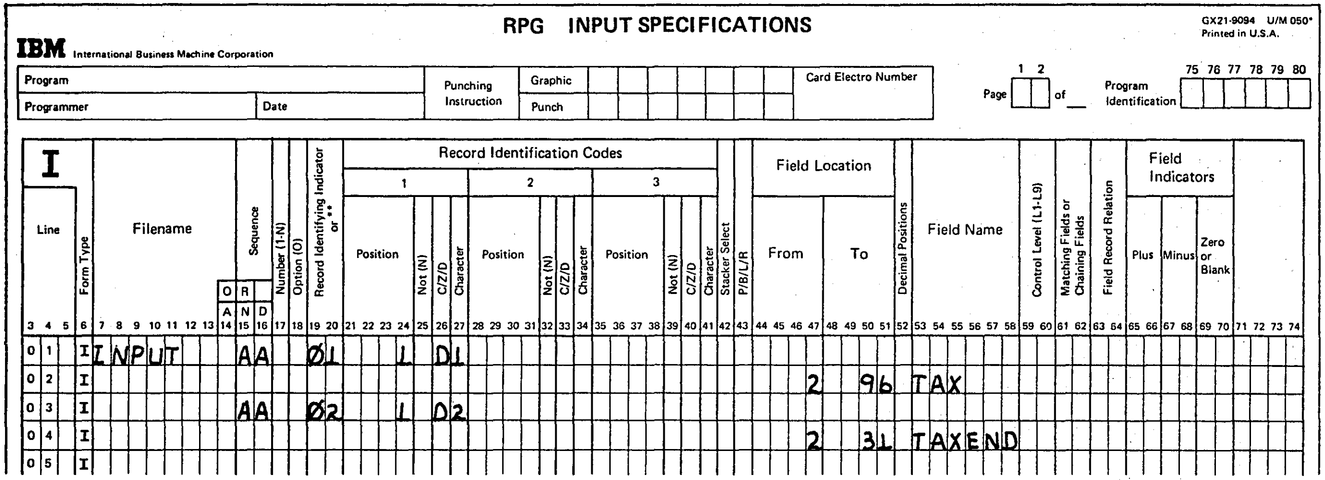
RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

IBM International Business Machine Corporation

Form X21-9091
Printed in U.S.A.

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
0 1	E				TAX		25	5									
0 2	E																

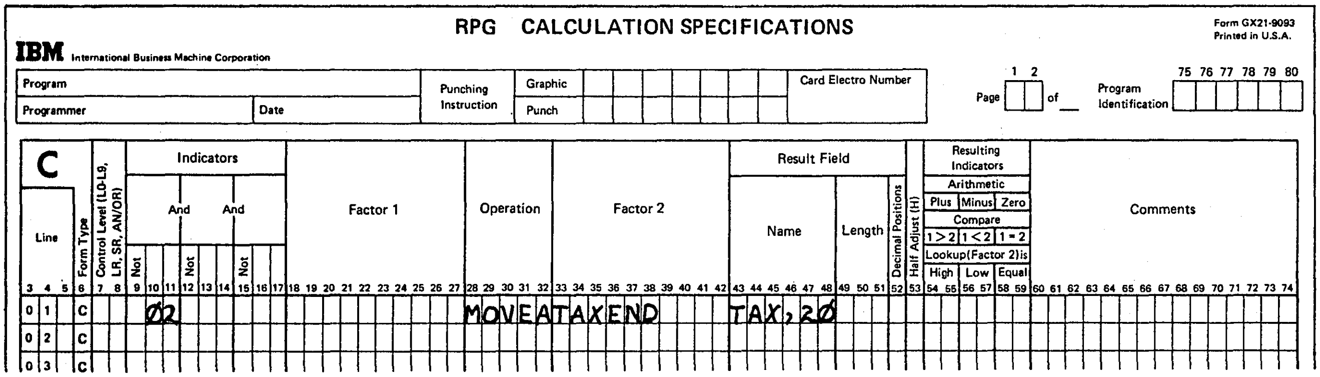


RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Field Location From	To	Decimal Positions	Field Name
0 1	I	INPUT	AA			Ø1	1		D	1				
0 2	I										2	96		TAX
0 3	I		AA			Ø2	1		D	2				
0 4	I										2	31		TAXEND
0 5	I													



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
0 1	C		Ø2		MOVEA	TAXEND	TAX,2Ø					
0 2	C											
0 3	C											

● Figure 9-49. Loading Array Data Consecutive on More Than One Record (Model 6, Model 10 Disk System, and Model 15)

Array Data Scattered on More Than One Record

Regardless of how many records are used to contain array data, if the fields are scattered on the records, each field must be individually loaded into its appropriate position in the array. However, a separate specification is not always necessary for each field of data to be loaded. In some cases, the same specification can be used for all the records. This depends on whether all the input records for a single array are organized in the same format and whether the fields from different records can be assigned the same name.

Assume that a 22-element array, named ARA, is defined. The data for the array is scattered on six input records, as shown in Figure 9-50. Although the array data is not consecutive, the four fields on each of the first five records are in the same format on each record. The remaining two fields on the sixth record are in the same format as the first two fields on all other records.

Since the array data follows the same organization on all records, describing one set of fields (Figure 9-51) actually describes the fields on all records, except the last. A separate input specification should be coded to indicate that record 6 only contains two of the fields. (Note on the Input sheet that records 1-5 are described in an OR relationship. Therefore, a specific card sequence cannot be specified in columns 15 and 16. You can assume that the array input records are in sequence. Record type 1 is the first record read and record type 6 is the last.)

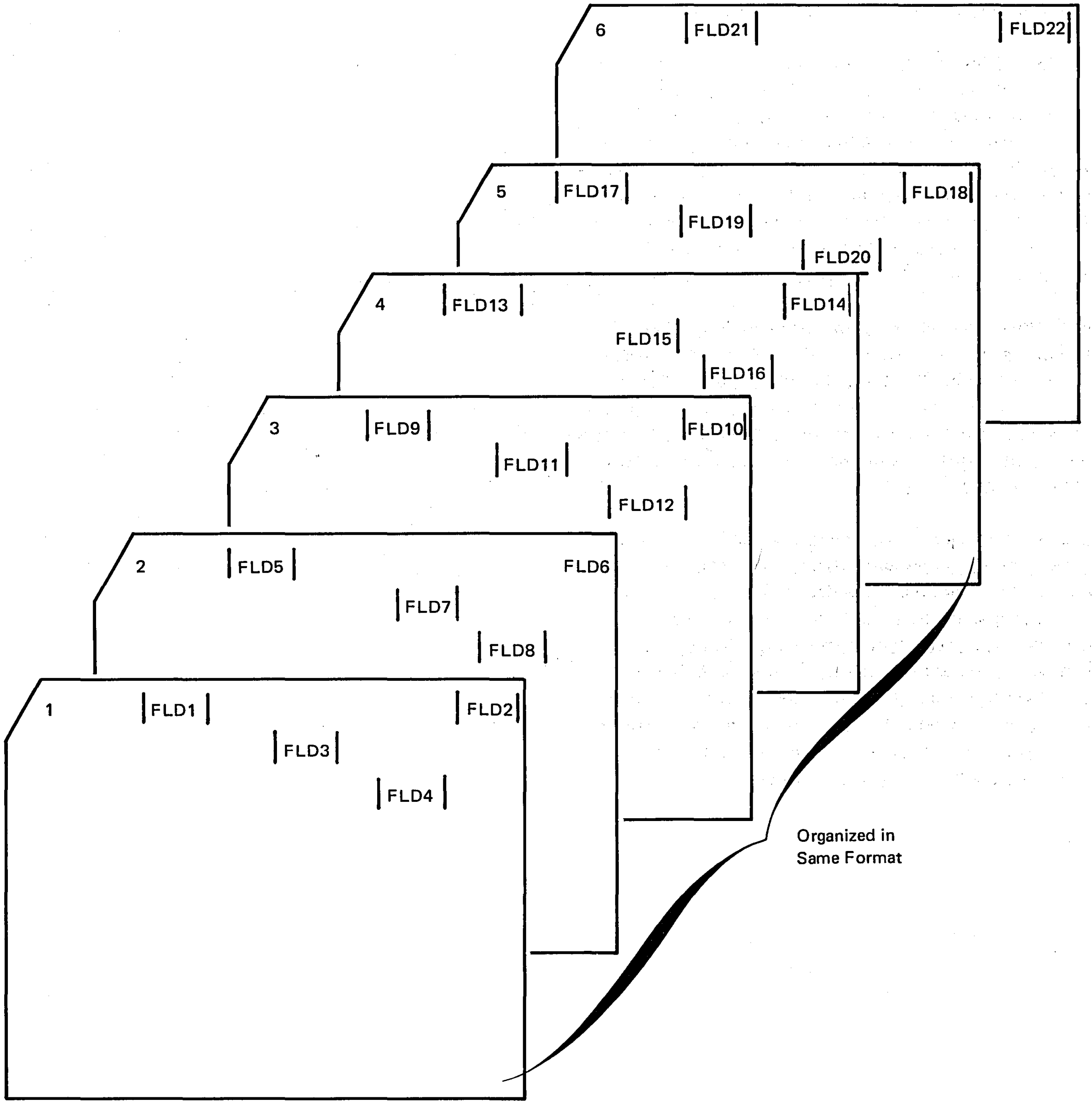
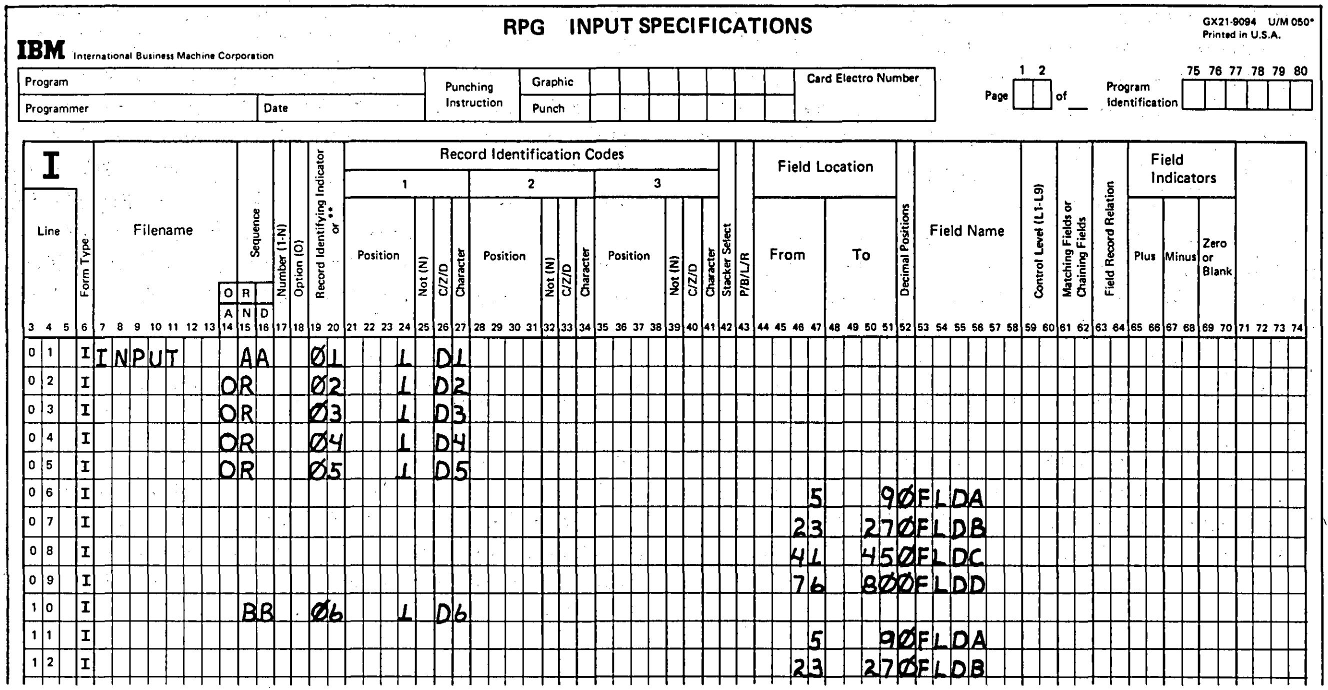


Figure 9-50. Array Data Scattered on More than One Record



RPG INPUT SPECIFICATIONS

IBM International Business Machine Corporation

GX21-9094 U/M 050* Printed in U.S.A.

Program

Programmer

Date

Punching Instruction

Graphic

Punch

Card Electro Number

Page 1 2 of

Program Identification 75 76 77 78 79 80

I
Line
Form Type
Filename
Sequence
OR
AND
Number (1-N)
Option (O)
Record Identifying Indicator or **
Record Identification Codes
1
2
3
Position
Not (N)
C/Z/D
Character
Stacker Select
P/B/L/R
Field Location
From
To
Decimal Positions
Field Name
Control Level (L1-L9)
Matching Fields or Chaining Fields
Field Record Relation
Field Indicators
Plus
Minus
Zero or Blank
3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74

```
01 IINPUT   AA  Ø1   1 D1
02 I       OR   Ø2   1 D2
03 I       OR   Ø3   1 D3
04 I       OR   Ø4   1 D4
05 I       OR   Ø5   1 D5
06 I                                        5   9ØFLDA
07 I                                       23  27ØFLDB
08 I                                       41  45ØFLDC
09 I                                       76  8ØØFLDD
10 I        BB  Ø6   1 D6
11 I                                        5   9ØFLDA
12 I                                       23  27ØFLDB
```

Figure 9-51. Describing One Set of Array Input Fields for Several Records

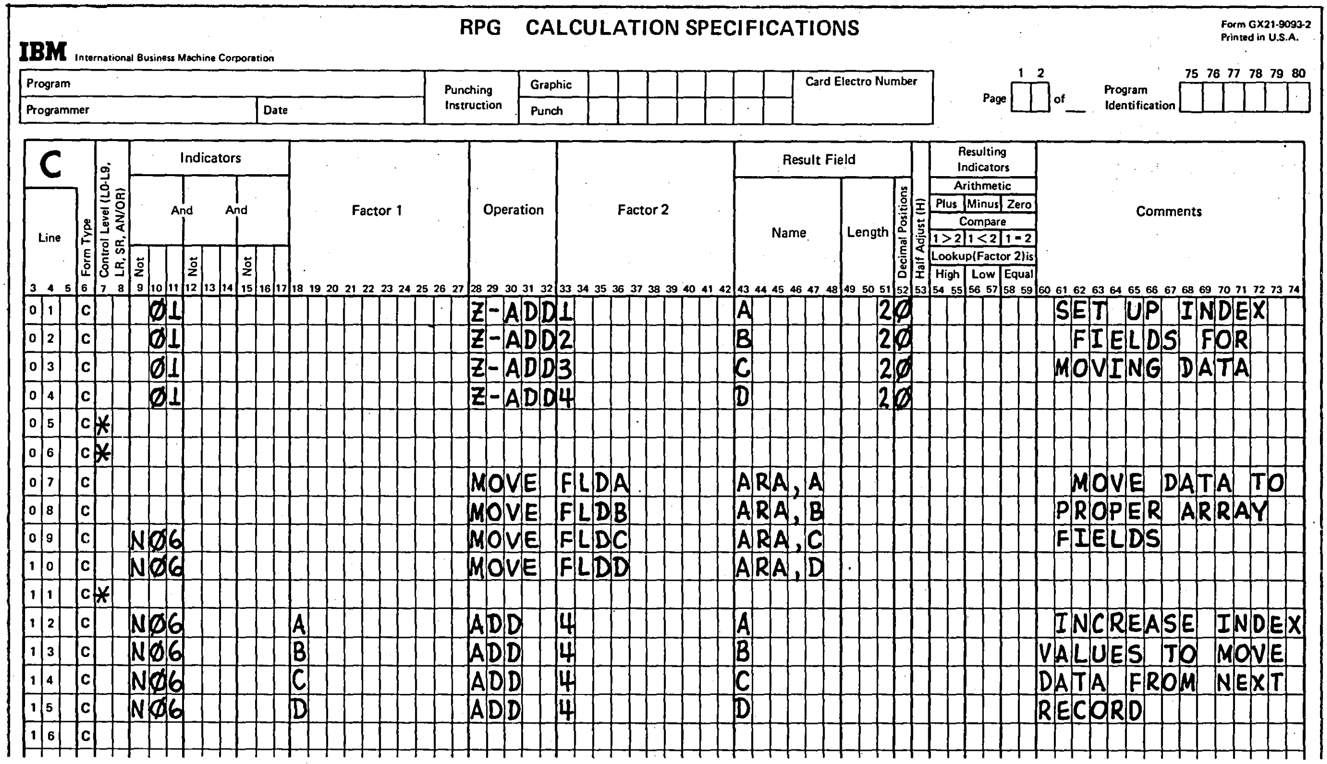
Because the fields on the different records have the same field names, only one MOVE specification is necessary for each unique field name. The specification on line 07 of Figure 9-52, when repeated for each record, moves FLDA of that record to the appropriate element of the ARA array. Lines 09 and 10 are performed for every record except the last, which does not have fields FLDC and FLDD.

Since the fields on the input records are in the same order as they are to be stored in ARA, a definite pattern is established as to where the data is to be moved. Fields from record 1 are stored in array elements one through four, fields from record 2 in array elements 5 through 8, fields from record 3 in array elements 9 through 12, and so on.

Array index fields can be used to indicate to which array elements that data is to be moved. For each unique field name, an individual index field should be set up. In this way, the values in the index fields only have to be changed every time another array input record is processed. When the first record is read, the index fields A, B, C, and D are initialized to 1, 2, 3, and 4, respectively, to prepare for moving the fields from record type 1 (Figure 9-52, lines 01-04). After the four fields are moved, the value 4 is added to each of the index fields so they point to where the four fields on the next record should be stored (Figure 9-52, lines 12-15). The same calculation specifications are repeated until fields FLDA and FLDB from the sixth record have been moved to the last two array elements.

Conditioning Operations Until All Array Data is Stored

All information must be stored in an array before you can reference the data by specifying the array name or array name with an index. Thus, any specifications to load the data into the array must be specified before any calculations which use the array information.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093-2
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Comments
01	C		Ø1		Z-ADD	1	A	2	Ø	SET UP INDEX
02	C		Ø1		Z-ADD	2	B	2	Ø	FIELDS FOR
03	C		Ø1		Z-ADD	3	C	2	Ø	MOVING DATA
04	C		Ø1		Z-ADD	4	D	2	Ø	
05	C	*								
06	C	*								
07	C				MOVE	FLDA	ARA,A			MOVE DATA TO
08	C				MOVE	FLDB	ARA,B			PROPER ARRAY
09	C		NØ6		MOVE	FLDC	ARA,C			FIELDS
10	C		NØ6		MOVE	FLDD	ARA,D			
11	C	*								
12	C		NØ6	A	ADD	4	A			INCREASE INDEX
13	C		NØ6	B	ADD	4	B			VALUES TO MOVE
14	C		NØ6	C	ADD	4	C			DATA FROM NEXT
15	C		NØ6	D	ADD	4	D			RECORD
16	C									

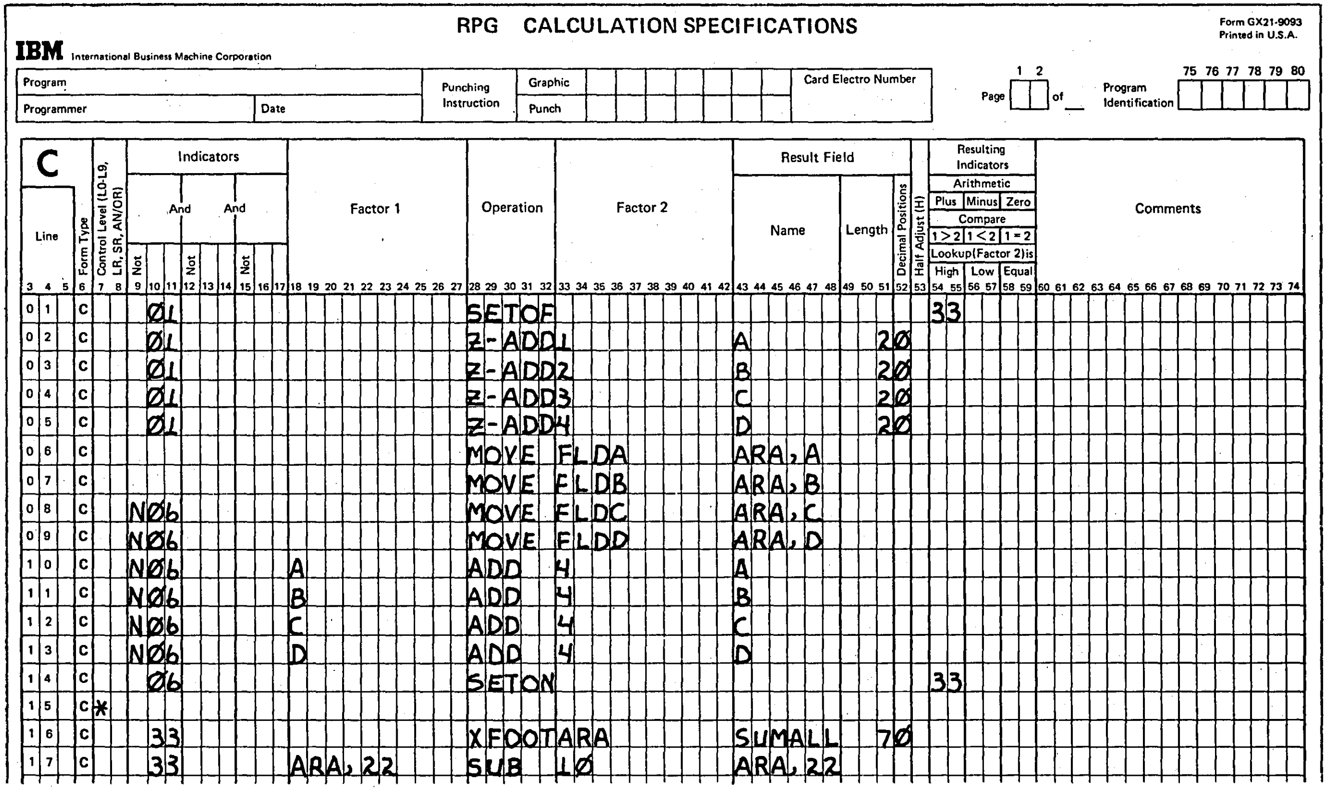
Figure 9-52. Using the Same MOVE for Fields from Several Records

For every record of array data, RPG II goes through a complete program cycle, just as it does to process any other data card. This means that input, calculation, and output specifications can be performed every time an array input record is processed. You want input specifications to be performed to describe the array record to the system and, perhaps, to load the array data at the same time. Perhaps, calculation specifications which move the data from the record to the array should also be performed. However, if there are still some array records which have not been processed (thus, not stored in the array), calculations and output which reference the array must not be performed. For example, if only five fields of data have been loaded into a 10-element array, adding all elements of the array or printing all elements will certainly not provide the results you want.

Once the last array input record has been stored, any specifications referencing the array elements can be performed. Thus, you must specify a conditioning indicator (columns 9 through 17 on Calculation sheet and columns 23 through 31 on Output sheet) which indicates when the last array record has been processed.

Lines 02 through 13 of the Calculation sheet in Figure 9-53 are performed to move data from the six array input records (see Figure 9-50) into the array ARA. When the last record is processed (record identifying indicator 06 on), the two array operations on lines 16 and 17 can be performed during that program cycle. Therefore, when record type 6 has been stored, indicator 33 is set on (Figure 9-53, line 14). Indicator 33 (or any other indicator which is set on) can then condition the XFOOT and SUB operations to be performed in a program cycle.

The record identifying indicator 06 was not specified to condition the array operations, because 06 is on only for the cycle in which the sixth array record is processed. Since the array operations on lines 16 and 17 must be performed in the following program cycles also (for example, if normal data records follow the array records), they must be conditioned by an indicator which is on during the following cycles. Once indicator 33 has been set on, it remains on through following program cycles, until set off (line 01) when another group of array records are processed.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

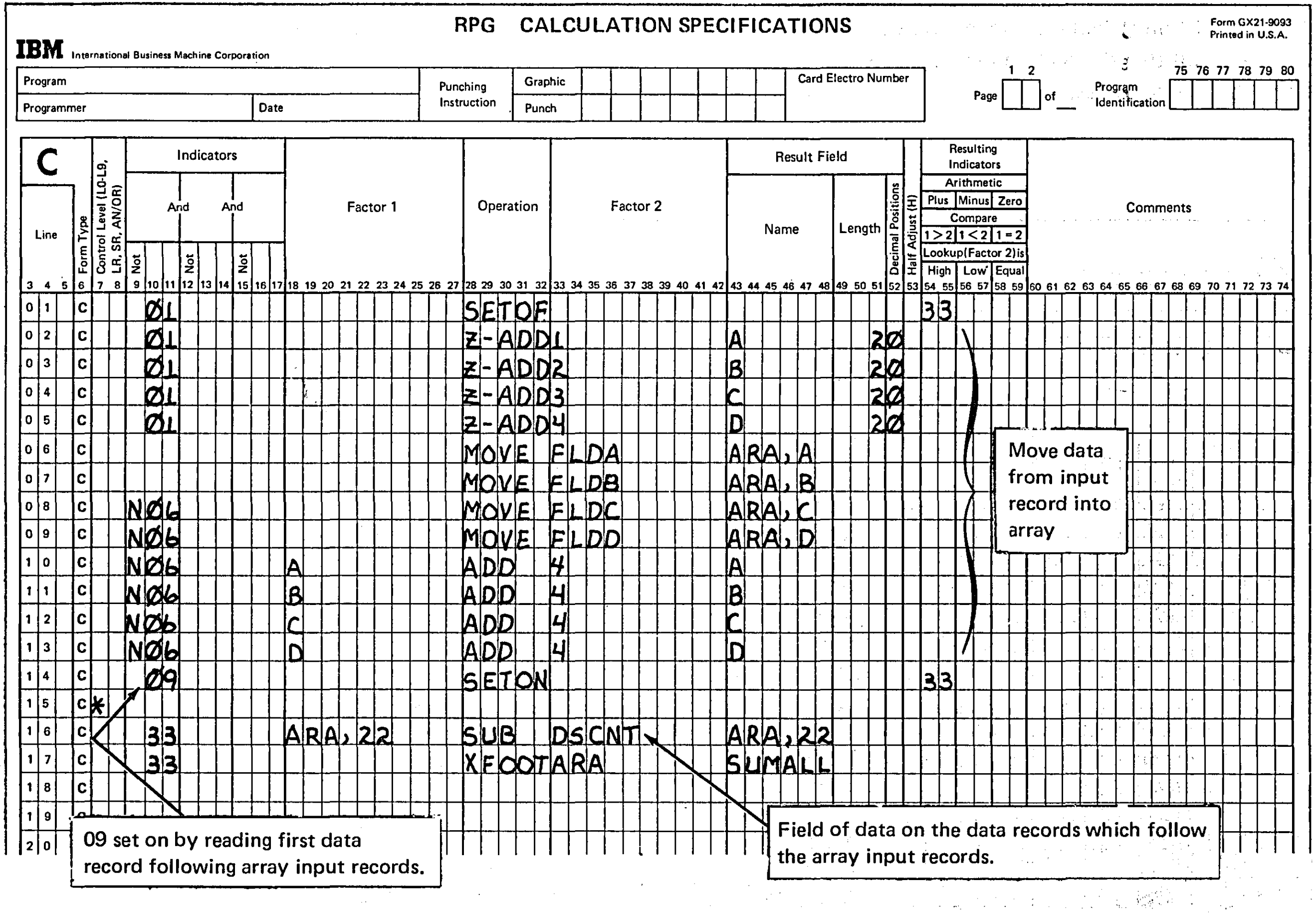
Program ____ Programmer ____ Date ____ Punching Instruction: Graphic ____ Punch ____ Card Electro Number ____ Page □□ of ___ Program Identification □□□□□□

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High / Plus / 1>2	Low / Minus / 1<2	Equal / Zero / 1=2	Comments
01	C		Ø1		SETOF						33			
02	C		Ø1		Z-ADD	1	A	2	Ø					
03	C		Ø1		Z-ADD	2	B	2	Ø					
04	C		Ø1		Z-ADD	3	C	2	Ø					
05	C		Ø1		Z-ADD	4	D	2	Ø					
06	C				MOVE	FLDA	ARA, A							
07	C				MOVE	FLDB	ARA, B							
08	C		NØ6		MOVE	FLDC	ARA, C							
09	C		NØ6		MOVE	FLDD	ARA, D							
10	C		NØ6	A	ADD	4	A							
11	C		NØ6	B	ADD	4	B							
12	C		NØ6	C	ADD	4	C							
13	C		NØ6	D	ADD	4	D							
14	C		Ø6		SETON						33			
15	C	*												
16	C		33		XFOOT	ARA	SUMALL	7	Ø					
17	C		33	ARA, 22	SUB	1Ø	ARA, 22							

Figure 9-53. Conditioning Operations Until All Array Data is Stored

Figure 9-54 shows how the array operations must be conditioned for another situation. In this case, record identifying indicator 06 does not set on indicator 33 because information (DSCNT) from data records following the array records must be available before the array operations can be performed (line 16). If indicator 06 caused indicator 33 to be set on, the array operations would be performed during the program cycle in which the sixth array record is stored. At that point, the DSCNT data is not available. Therefore, record identifying indicator 09 (the first type of data card following the array records) sets on the conditioning indicator 33 instead.

At this point, we must mention a problem which can come up if array elements are contained on more than one record (or the same record type), and the records contain normal input data as well as array data. Assume three cards contain the array data and all the data must be stored in the array prior to performing any calculation or output operations. This means the three records must be read before processing. As each new record (of the same record type) is read, the data from the previous record is destroyed, unless it has been moved or stored in a special place, such as an array. Since normal input data (nonarray fields) from the first two records is no longer available once the third record has been read, any calculation or output specifications which reference this input data might give incorrect results.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page of | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High	Low	Equal	Comments
01	C		Ø1		SETOF						33			
02	C		Ø1		Z-ADD	1	A	2	Ø					
03	C		Ø1		Z-ADD	2	B	2	Ø					
04	C		Ø1		Z-ADD	3	C	2	Ø					
05	C		Ø1		Z-ADD	4	D	2	Ø					
06	C				MOVE	FLDA	ARA,A							
07	C				MOVE	FLDB	ARA,B							
08	C		NØ6		MOVE	FLDC	ARA,C							
09	C		NØ6		MOVE	FLDD	ARA,D							
10	C		NØ6	A	ADD	4	A							
11	C		NØ6	B	ADD	4	B							
12	C		NØ6	C	ADD	4	C							
13	C		NØ6	D	ADD	4	D							
14	C		Ø9		SETON						33			
15	C	*												
16	C		33	ARA,22	SUB	DSCNT	ARA,22							
17	C		33		XFOOT	ARA	SUMALL							
18	C													
19	C													
20														

Move data from input record into array

09 set on by reading first data record following array input records.

Field of data on the data records which follow the array input records.

Figure 9-54. Conditioning Operations Until All Array Data is Stored and Input Data is Available

Review 9

1. In which of the following ways is an array like a table (state true or false and the reasons for your answer)?
   a. Each can be referenced as one group of information.
   b. Each is a continuous series of data fields (elements) stored side by side.
   c. A particular item of data can be individually referenced in either a table or an array.
   d. Each is defined by coding extension specifications.

2. Can one array be compared to another array to determine which is greater or less? State the reason for your answer.

3. Explain what happens if an array *(a)* of 18 elements is added to an array *(b)* of three elements, with the result placed in an array *(c)* of 18 elements.

4. The following array (ARASIX) is to be set up during a program run:

1	2	72	5	20	15

   a. Define the array on an Extension sheet.
   b. If ARASIX is multiplied by *3*, what data will be placed in the result array RESARA?
   c. Should the result array RESARA be defined on the Extension sheet also?
   d. If so, code the necessary extension specifications to define RESARA.
   e. What is accomplished by defining an array on the Extension sheet?

5. a. Explain what happens when an XFOOT operation is performed.
   b. If ARASIX (refer to question 3) is specified on the Calculation sheet as Factor 2 of an XFOOT operation, what data would be placed in the result field?

6. How does a programmer specify that an entire array is to be printed or punched (a) during output time in the object program cycle; (b) at end of job?

7. How does a programmer specify whether an entire array or only a particular array element is to be operated upon or used for output?

8. Data for a SALES array is recorded on a one-record input file called INFILE in the following format:

Field	*Columns*	*Field*	*Columns*
Clerk1	1-10	Clerk6	51-6u
Clerk2	11-20	Clerk7	61-70
Clerk3	21-30	Clerk8	71-80
Clerk4	31-40	Clerk9	81-90
Clerk5	41-50	record code	91 (not array data)

Each of the clerk fields contains data with two decimal positions. Code the specifications necessary to:

a. Define the array as a pre-execution time array.

b. If necessary, describe the input record and store the data in the SALES array.

9. Show two ways that data from the first five fields of the following record could be stored in an execution time array named SET (15 elements, three numeric characters each, no decimal positions). Column 1 of the record contains a P as the record identifying code.

Field	Columns	Field	Columns	Field	Columns
Fld1	2-4	Fld6	22-24	Fld11	42-44
Fld2	6-8	Fld7	26-28	Fld12	46-48
Fld3	10-12	Fld8	30-32	Fld13	50-52
Fld4	14-16	Fld9	34-36	Fld14	54-56
Fld5	18-20	Fld10	38-40	Fld15	58-60

10. a. Code the output specifications to print the 13th element of the array SET (output filename PRINT).

b. Code the specifications to print the entire array SET at end of job (output filename PRINT).

11. SEARCH is the name of a field containing data you wish to locate in the 6-element PAY array. Code the specifications to search the array to determine if the data is present. If present, print the number and contents of the array element. If more than one array element satisfies the search, each is to be printed.

12. Code specifications to:

a. Add ARA1 to ARA2 and place the result in ARA2.

b. Sum all elements of ARA2 and place the result in TOTAL.

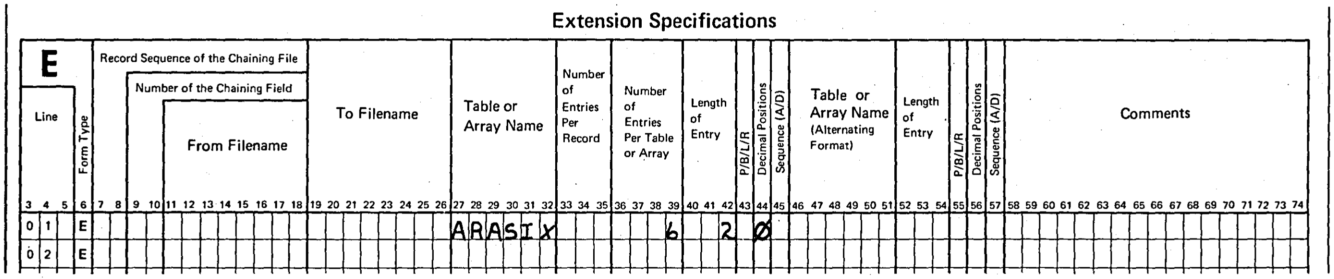
c. Print both results.

1. a. False. Only one table element can be referenced at one time.
   b. True.
   c. True.
   d. True.

2. No. An operation to be performed on an array is repeated for each element in the array. Therefore, a compare (COMP) operation cannot give a meaningful result for the entire array. The two arrays, however, could be totaled using the XFOOT operation and the resulting totals could be compared.

3. The first three elements of array *a* would be added to the three elements of array *b*, with the three results placed in the first three elements of array *c*. The remaining 15 elements of array *c* (result array) remain unchanged.

4. a. Entries shown are required; no other entries should be made. The entry in column 44 is necessary to indicate the elements are numeric for arithmetic operations.



Extension Specifications

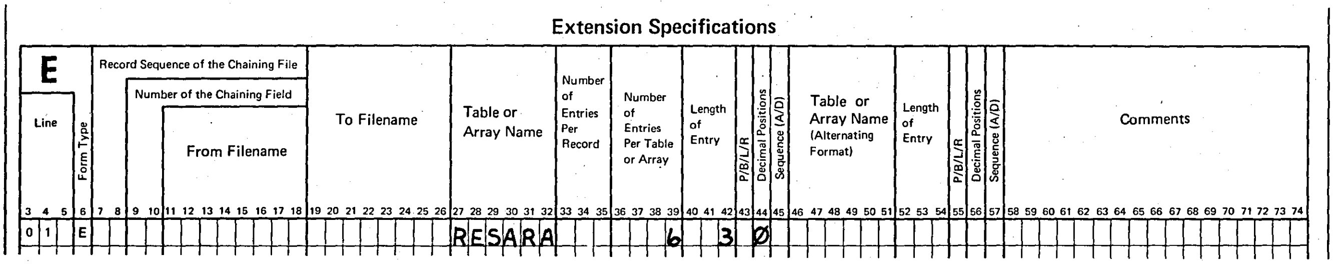
E Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					ARASIX		6	2		Ø							
02	E																	

b.

3	6	216	15	60	45

c. Yes, all arrays to be used in a program must be defined on the Extension sheet.

d. Entries shown are required; no other entries should be made. Length of array element (columns 40-42) must be 3 to contain the largest addition result.



Extension Specifications

E Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					RESARA		6	3		Ø							

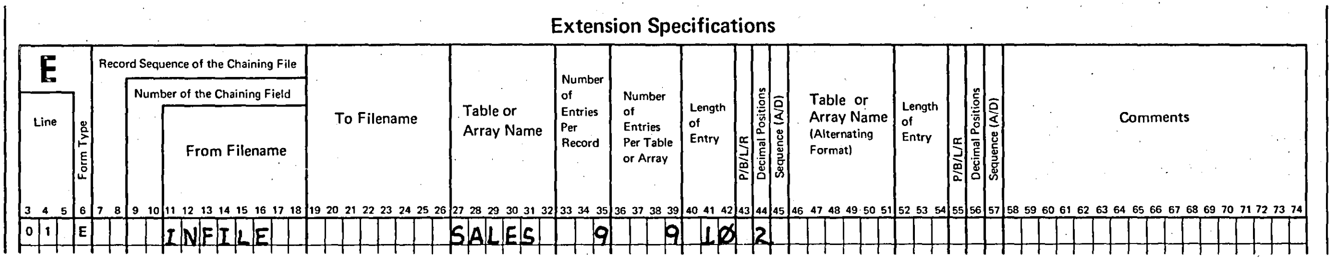
e. An area in storage sufficient to contain the array data is reserved. The actual array data may be stored in the array later, when input records are read or during calculations.

5. a. The XFOOT operation causes all elements in the array specified as Factor 2 to be added together. The single result of the additions is placed in the result field specified with the XFOOT operation.
   b. The total of all elements in the ARASIX array (115) would be placed in the result field.

6. a. The name of the array is specified under Field Name (columns 32 through 37) on the Output sheet. The filename (columns 7 through 14) must also be specified, as for output of any field.
   b. The name of the output file is specified under To Filename (columns 19-26) on the Extension sheet. No entry is necessary on the Output sheet. (This method of array output cannot be used for execution time arrays.)

7. The array name is specified alone (on the Calculation or Output sheet) to reference the entire array. The array name must be followed by a comma and an index number or index field to reference only a particular array element.

8. a. Extension specifications to define the array:

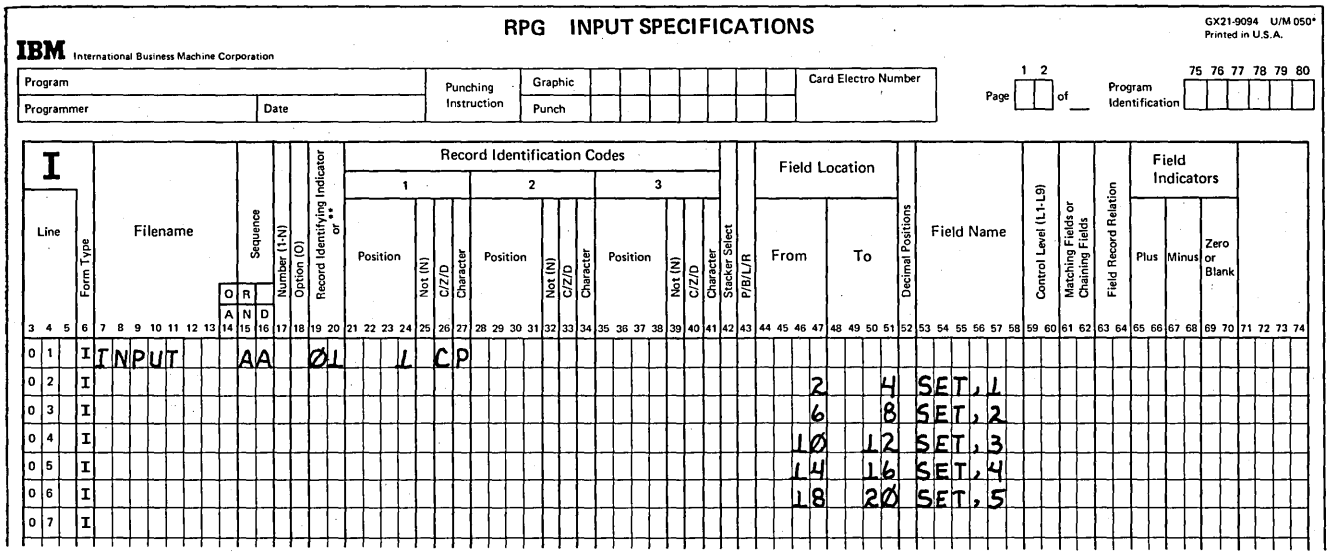


Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P/B/L/R (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P/B/L/R (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E			INFILE		SALES	9	9	1Ø		2							

b. Input specifications to describe the input record and store the data in the SALES array are not necessary, since the array is automatically loaded before execution of the program.

9. First method:



RPG INPUT SPECIFICATIONS

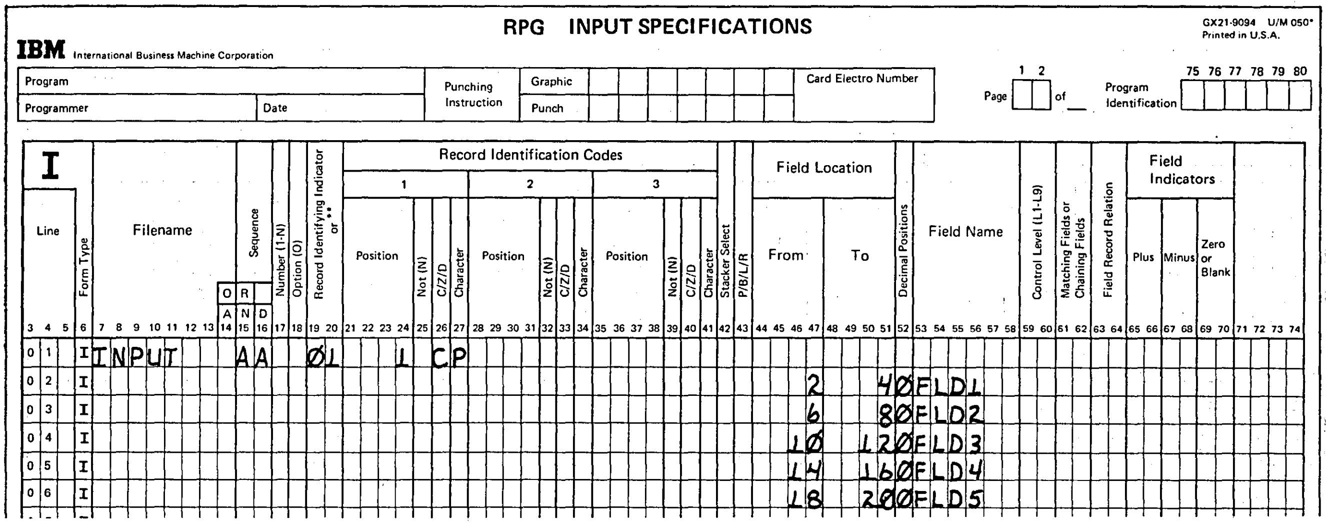
IBM International Business Machine Corporation

GX21-9094 U/M 050*
Printed in U.S.A.

Program | Programmer | Date | Punching Instruction: Graphic, Punch | Card Electro Number | Page (1 2) of | Program Identification (75-80)

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)
01	I	INPUT	AA			Ø1	1		C	P										
02	I										2	4		SET,1						
03	I										6	8		SET,2						
04	I										1Ø	12		SET,3						
05	I										14	16		SET,4						
06	I										18	2Ø		SET,5						
07	I																			

Second method:

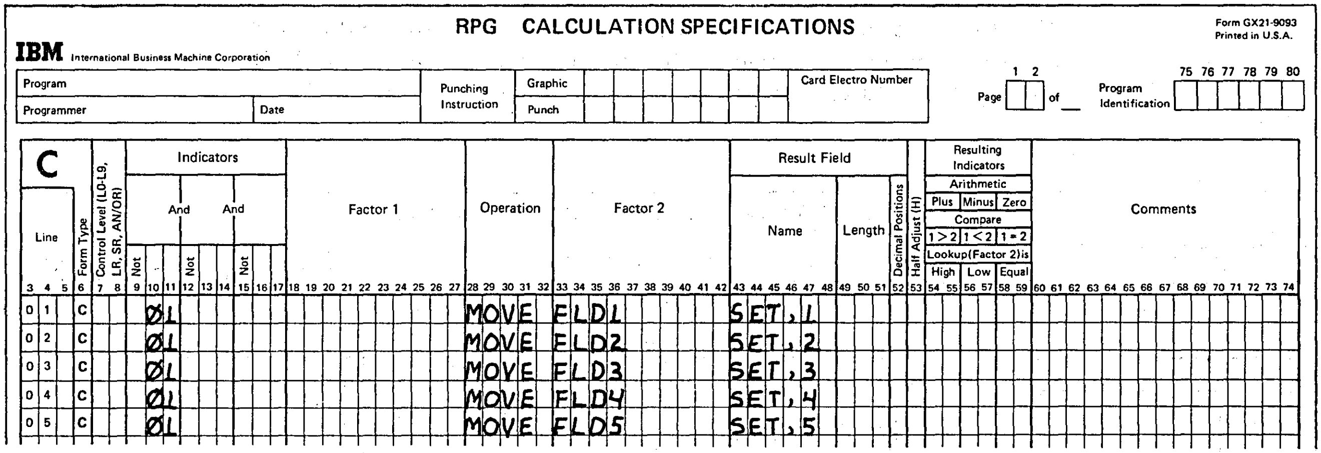


IBM International Business Machine Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	C/Z/D	Character	From	To	Decimal Positions	Field Name
01	I	INPUT	AA	Ø1	1	C	P				
02	I							2	4	Ø	FLD1
03	I							6	8	Ø	FLD2
04	I							1Ø	12	Ø	FLD3
05	I							14	16	Ø	FLD4
06	I							18	2Ø	Ø	FLD5



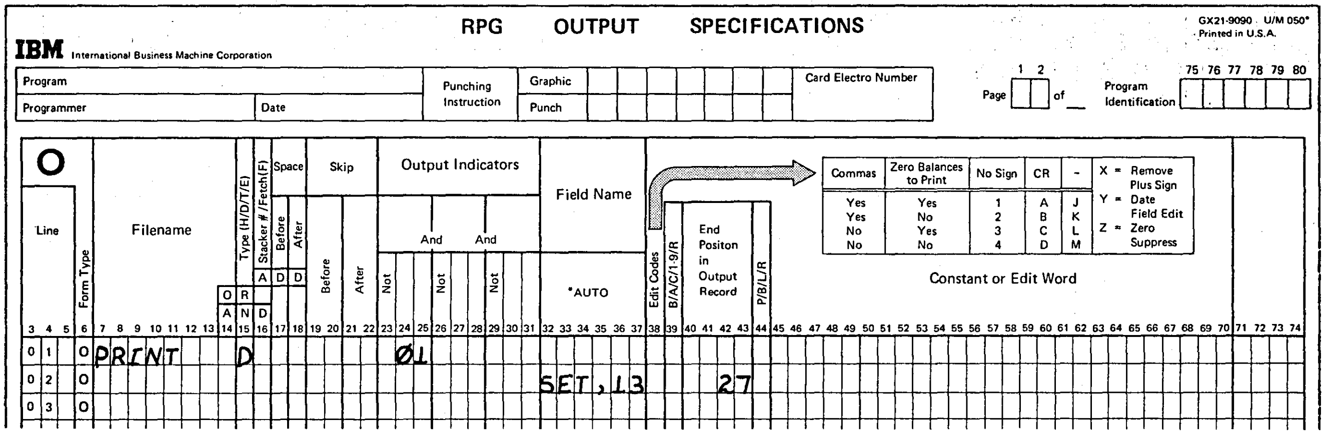
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name
01	C	Ø1		MOVE	FLD1	SET,1
02	C	Ø1		MOVE	FLD2	SET,2
03	C	Ø1		MOVE	FLD3	SET,3
04	C	Ø1		MOVE	FLD4	SET,4
05	C	Ø1		MOVE	FLD5	SET,5

10. a. Output of 13th element of SET array:



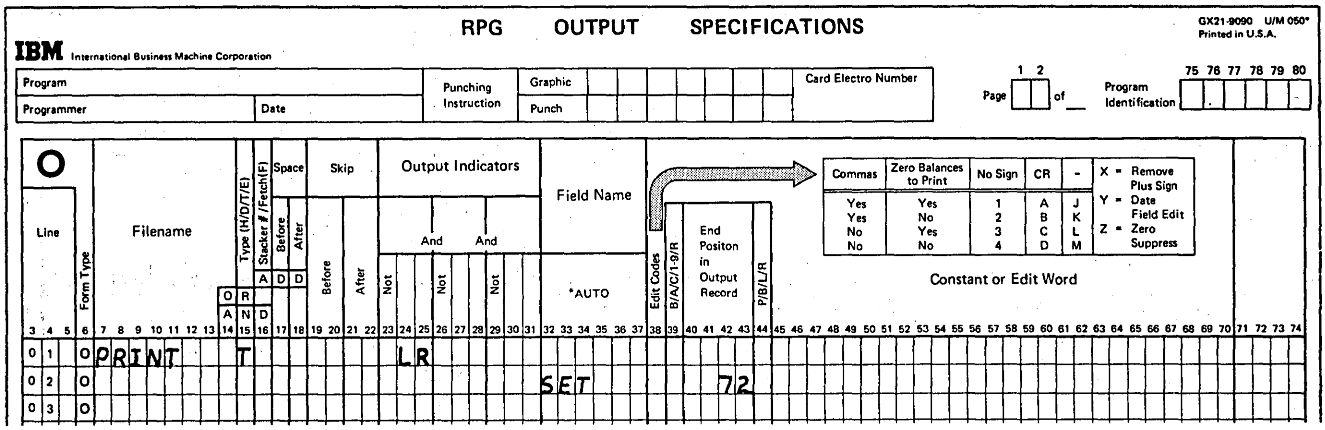
IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record
01	O	PRINT	D	Ø1		
02	O				SET,13	27
03	O					

b. Output of entire SET array at end of job since SET is an execution time array, must be done using output specifications, as shown below:



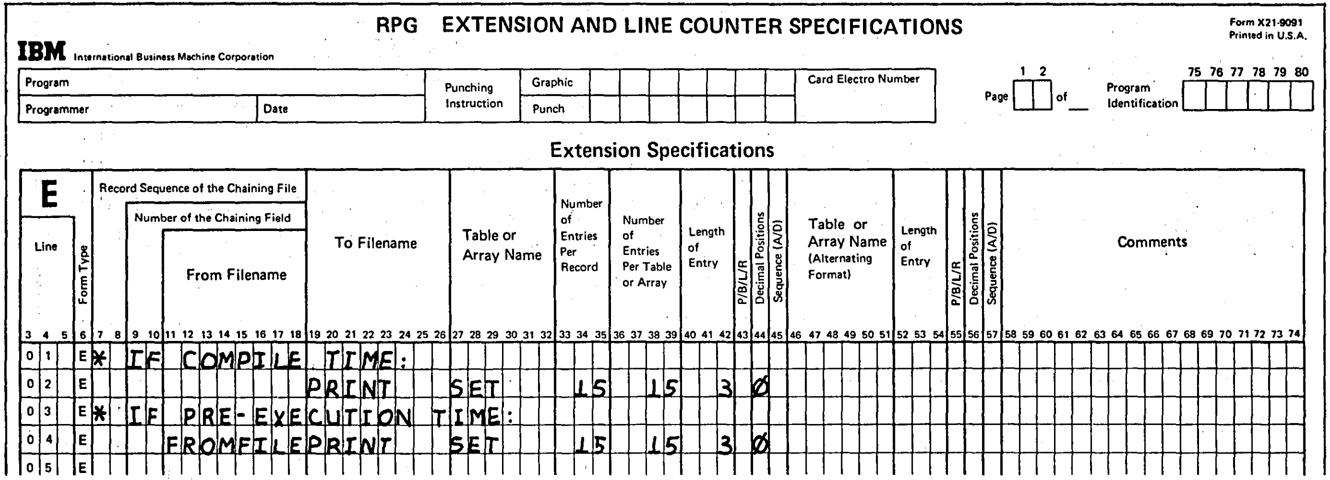
IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record
01	O	PRINT	T	LR		
02	O				SET	72
03	O					

If SET were a compile time array or a pre-execution time array, output of the entire array could be accomplished by entering the output filename in To Filename (columns 19-26) in the extension specification for the array:



IBM International Business Machine Corporation

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091 Printed in U.S.A.

Extension Specifications

Line	Form Type	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	Decimal Positions
01	E	* IF COMPILE TIME:						
02	E		PRINT	SET	15	15	3	Ø
03	E	* IF PRE-EXECUTION TIME:						
04	E	FROMFILE	PRINT	SET	15	15	3	Ø
05	E							

11.

RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Form GX21-9093
Printed in U.S.A.

Program		Punching Instruction	Graphic		Card Electro Number	Page	of	Program Identification
Programmer	Date		Punch					

Line	Form Type	Control Level	Indicators Not	Indicators	Not	And	Not	And	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	High 1>2 (Plus)	Low 1<2 (Minus)	Equal 1=2 (Zero)	Comments
01	C									Z-ADD	1	IX	1	Ø					
02	C								AGAIN	TAG									
03	C								SEARCH	LOKUP	PAY,IX							23	FOUND, 23 ON
04	C			23						EXCPT									
05	C			23					IX	ADD	1	IX							
06	C			23					IX	COMP	7						24		
07	C			24		23				GOTO	AGAIN								
08	C																		

RPG OUTPUT SPECIFICATIONS

IBM International Business Machine Corporation

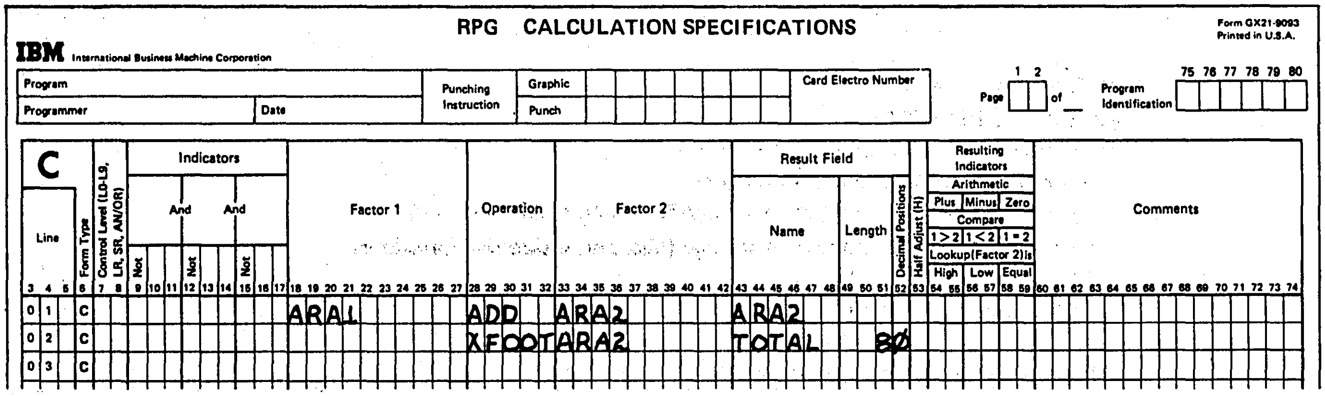
GX21-9090 U/M 050*
Printed in U.S.A.

Program		Punching Instruction	Graphic		Card Electro Number	Page	of	Program Identification
Programmer	Date		Punch					

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Stacker	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	PRINT	E						23						
02	O									IX			5		
03	O									PAY,IX			74		
04	O														

12.



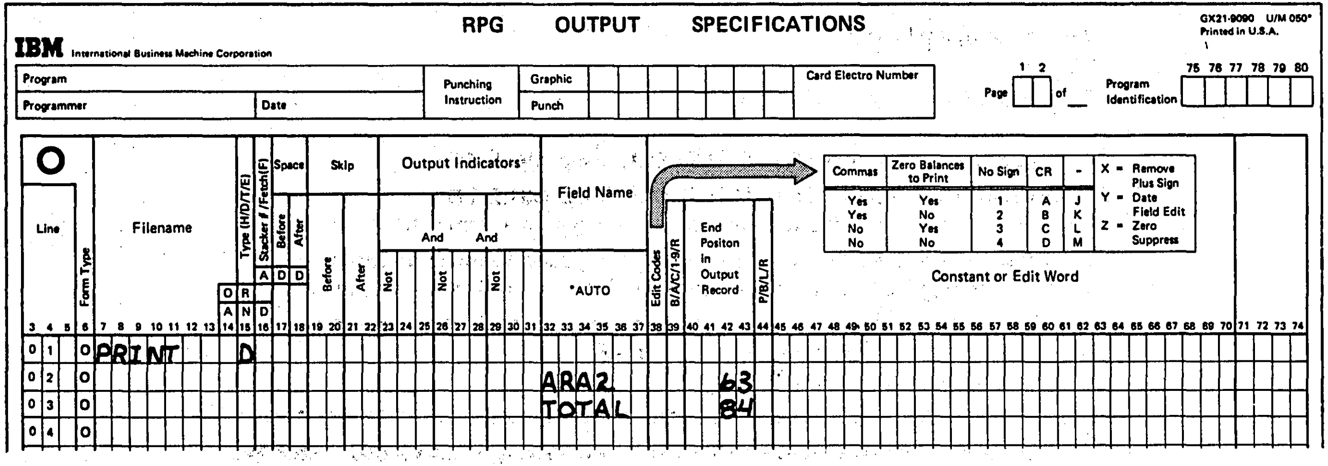
IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page of
Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C			ARA1	ADD	ARA2	ARA2					
02	C				XFOOT	ARA2	TOTAL	8	Ø			
03	C											



IBM International Business Machine Corporation

RPG OUTPUT SPECIFICATIONS

GX21-9090 U/M 050*
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction
Graphic
Punch
Card Electro Number
Page of
Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker #/Fetch(F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	PRINT	D												
02	O									ARA2			63		
03	O									TOTAL			84		
04	O														

Chapter 10. Working With Data Structure

CHAPTER 10 DESCRIBES:

Representation of characters on cards.

Representation of characters in storage (disk and inside the computer).

Packed and binary data.

Collating sequence of characters.

Move zone operations.

File translation.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe the representation of characters and negative numbers on cards.

Describe the representation of characters on disk and inside the computer.

Define byte, bit, zone portion and digit portion.

Compare the storing of characters on cards to the storing of characters in storage.

Identify bit combinations with numerical values.

Assign numerical values to zone and digit portions.

Define unpacked decimal format, packed decimal format, and binary format.

Describe the hexadecimal numbering system.

Describe the collating sequence of characters.

Code specifications to change the collating sequence.

Alter the structure of characters in storage by using move zone operations.

Translate characters by coding the Translation Table and Alternate Collating Sheet.

Note: You can use the review questions contained in ***Review 10*** at the end of this chapter to test your comprehension of each topic in the chapter. Questions are grouped according to the topic to which they apply. Answers follow the review questions.

CHARACTER STRUCTURE

Representation of Characters on 96-Column Cards

Punched cards provide data the computer is to work with. Each of the 96 columns of a card can contain punches for a single character. Therefore, up to 96 characters of information can be represented on a single record.

Each column of a card consists of six punch positions, labeled *B, A, 8, 4, 2,* and *1,* from the top of a column to the bottom. Characters are represented by a combination of from zero to six holes punched in the punch positions of a single column.

Since there are six punch positions available, the number and positions of the holes may be varied to form 64 different punch combinations. Each unique combination of punches is associated with a particular character. Therefore, you can represent any one of 64 different characters in a card column (Figure 10-1).

A card column consists of both a zone portion and a digit portion. *B* and *A* are referred to as zone punch positions, while *8, 4, 2,* and *1* are digit punch positions. The combinations of zone and digit punches make it possible to separate the characters into three groups (Figure 10-1):

- Alphabetic letters are represented by at least one punch in both the zone and digit portions of a column.
- The 28 special characters can consist of no punches, only zone punches, only digit punches, or both zone and digit punches.
- Positive numbers are represented by holes only in the digit punch positions. The one exception is the number *0* which is represented by a single punch in the *A* zone punch position.

			Numeric Characters									
			0	1	2	3	4	5	6	7	8	9
Punch Positions	Zone	B										
		A	A									
	Digit	8									8	8
		4					4	4	4	4		
		2			2	2			2	2		
		1		1		1		1		1		1

			Alphabetic Characters																									
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Punch Positions	Zone	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B								
		A	A	A	A	A	A	A	A	A	A										A	A	A	A	A	A	A	A
	Digit	8								8	8								8	8							8	8
		4				4	4	4	4						4	4	4	4					4	4	4	4		
		2		2	2			2	2				2	2			2	2			2	2			2	2		
		1	1		1		1		1		1	1		1		1		1		1		1		1		1		1

			Special Characters																									
			}	¢	.	<	(	+	|	!	$	*	)	;	¬	-	/	&	,	%	–	>	?	:	#	@	'	=	"	b̸
Punch Positions	Zone	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B														
		A	A	A	A	A	A	A	A								A	A	A	A	A	A	A							
	Digit	8		8	8	8	8	8	8	8	8	8	8	8	8			8	8	8	8	8	8	8	8	8	8	8	8	
		4				4	4	4	4			4	4	4	4					4	4	4	4			4	4	4	4	
		2		2	2			2	2	2	2			2	2			2	2			2	2	2	2			2	2	
		1			1		1		1		1		1		1		1		1		1		1		1		1		1	

51708

Figure 10-1. Character Set and Punch Combinations

Representation of Negative Numbers

Note that *positive* numbers are represented only by digit punches. Negative numbers (*-1* through *-9* and *-0*) can also be represented to the computer. However, to indicate that a number is negative, a column must contain both the punch combination for the number and the punch combination for the minus sign. As column 7 of Figure 10-2 shows, the *8* and *1* digit-punch positions are punched to represent the number *9.* A *-9* is represented in column 12 by the same digit punches plus a hole in the *B* zone-punch position.

As mentioned, all 64 possible punch combinations are associated with a character. Therefore, adding a *B* zone punch to the punch combination of a number means the punch combination for any negative number is the same as the punch combination already assigned to one of the 64 characters. The negative numbers *-1* through *-9* are represented by the same punch combinations as the letters *J* through *R*; -0 has the same punch combination as the special character }

Note: Although the value -0 (negative zero) is not used by itself, it can exist as a punch combination in the units position of a data field.

The RPG II program determines whether the punch combination is a letter or a number according to whether an entry has been made in column 52 of the input specifications. Column 52 is used to specify the number of decimal positions in a field. If an entry is present, the RPG II program assumes any character in that field to be numeric. Absence of an entry in column 52 tells the RPG II program that it is reading either a letter or a special character in an

CHARACTERS

Punch Position	–1	–2	–3	–4	–5	–6	–7	–8	–9	–0
B	●	●	●	●	●	●	●	●	●	●
A										●
8								●	●	
4				●	●	●	●			
2		●	●			●	●			
1	●		●		●		●		●	

Punch Position	J	K	L	M	N	O	P	Q	R	}
B	●	●	●	●	●	●	●	●	●	●
A										●
8								●	●	
4				●	●	●	●			
2		●	●			●	●			
1	●		●		●		●		●	

● Figure 10-3. Negative Number Punch Combinations the Same as Punch Combinations for J-R

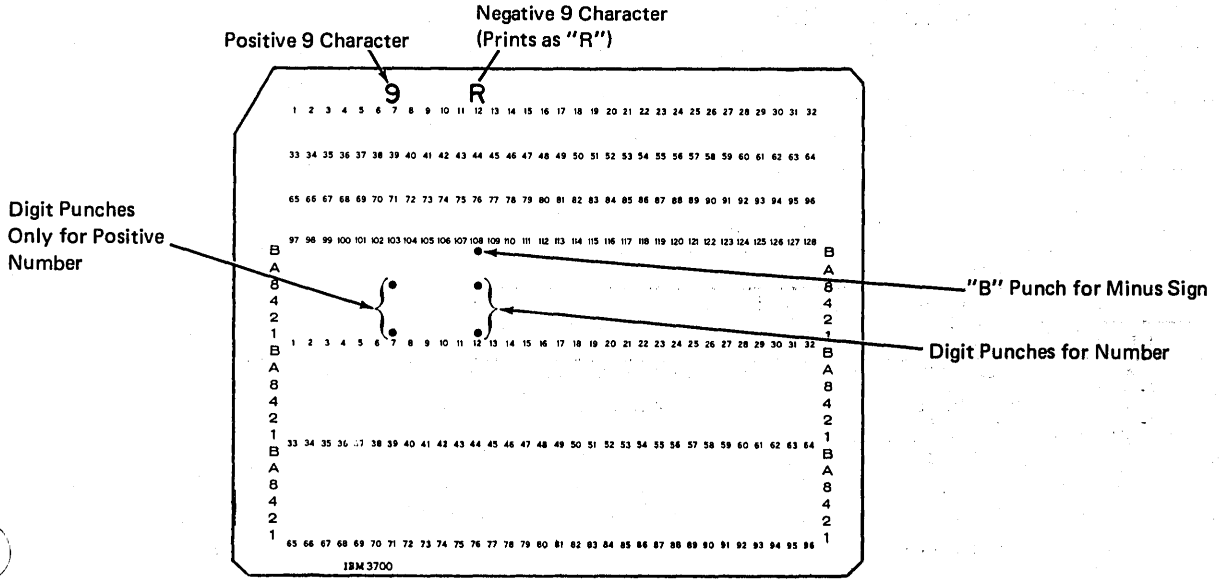


Figure 10-2. Punches for Negative Numbers

alphameric field. By examining column 52, the RPG II program recognizes when the *B* zone punch is associated with the punch combination of one of the letters *J* through *R* or the punch combination of a negative number.

In the discussion so far, you have learned how data is recorded in a form which the computer can understand. The data is represented as punched holes on 96-column cards. Before the RPG II program can use the data, as in calculations or output operations, it must store the information. The data is then available in computer storage whenever it is needed during the run of a program.

Representation of Characters in Storage

When you look at the punch area of a card, you cannot immediately determine which characters are stored on the card. First, you have to determine which character is associated with a particular punch combination. The punched holes, then, are the means of representing characters on a card. Similarly, a character such as the letter *A* is not stored on disk or inside the computer in a form you would recognize as the letter *A*. On disk or inside the computer, there is also a means of representing characters.

Information from each of the 96 columns of punched cards can be transferred to disk. Data from each column is stored in corresponding positions on disk in the form of magnetized spots.

Characters are represented electronically in computer storage. The storage area of the computer consists of a number of magnetic *bits*, which can be turned on or off by passing an electric current through them. The exact details of how this is done is not important to this discussion; what is important is that each bit can be in either an on state or an off state. We use a *1* to show a bit that is on; while a *0* represents a bit that is off (Figure 10-4).

The magnetic bits inside the computer or on disk are arranged in groups, called *bytes,* just as the punch positions on a card are arranged in groups called columns (Figure 10-4). Just as each column on a card can contain a character, each byte in storage can also contain a character. A particular combination of on and off bits in a byte represent a certain character inside the computer, just as a particular combination of punched and unpunched positions in a column represent that character on a card.

Data is represented on a card, character by character; likewise, data is stored inside the computer, character by character. Just as you can look at a punched card and refer to a character by the particular column containing that character's punch combination, the computer can reference a character by the particular byte in storage which contains that character's bit combination.

Difference Between Character Representation on Cards and in Storage

Although there are many similarities in the way a character is represented in storage and on a punched card, it is important to note one difference. While a card column consists of six positions, a byte consists of eight positions or bits. Thus, within the computer, eight positions are used to represent a single character, whereas only six positions are available on a card.

A byte is divided into a zone portion and a digit portion, just as a card column is (Figure 10-5). The four digit positions, in both a byte and a column, are labeled *1, 2, 4,* and *8.* However, a byte contains four zone positions, whereas a card column contains only two zone positions.

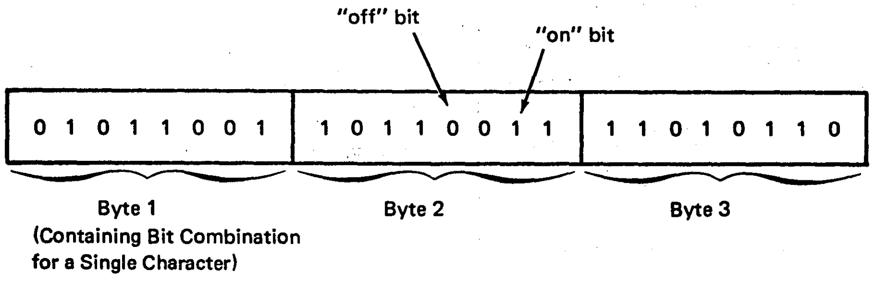


Figure 10-4. Representation of Characters in Storage

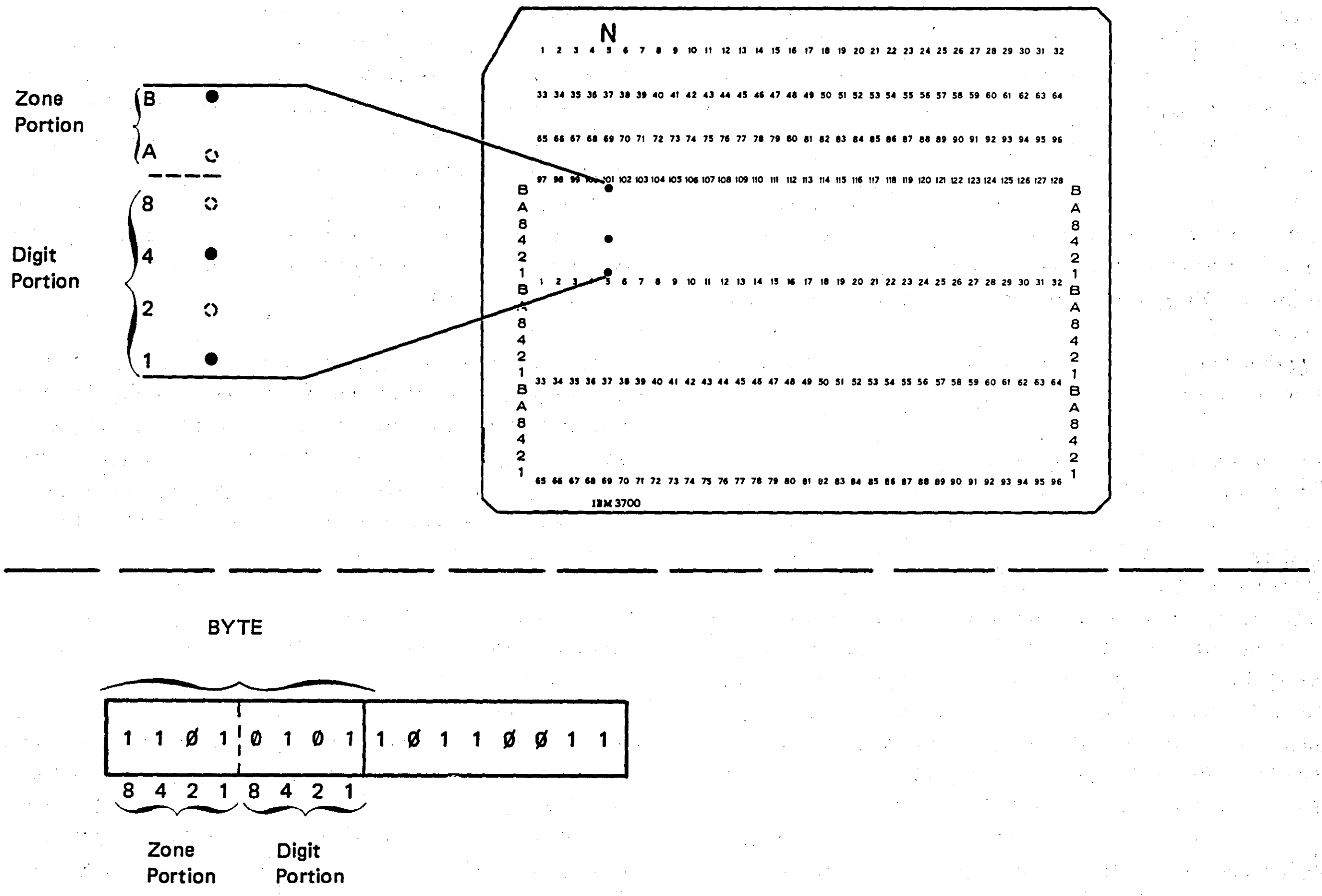


Figure 10-5. Correspondence Between a Byte and a Card Column

Since there are four digit positions in botn a byte and a card column, the digit portion of a byte corresponds one-for-one with the digit portion of that character's punch combination. That is, if a digit punch position is punched, the corresponding digit bit is set on *(1)* in storage. Likewise a digit bit is set off *(0)* if the corresponding punch position does not contain a punch. To check this, note how the digit portion of the plus sign (+) character is represented on a card and in storage (Figure 10-6). (*Note:* Ampersand (&) is an exception to this rule; see Figure 10-7.)

The zone portions of a card column and a byte do not correspond one-to-one, however. This is because there are four zone bits in storage for each character, while there are only two zone positions in a card column. Looking at Figure 10-6 again, you can see that even though *A* and *B* punch positions contain punches for the plus sign character, the 2 and 1 zone bits in storage are not on.

Since the zone portions differ, a translation takes place when a card is read and the data (characters) is stored inside the computer. The machine reads the punch combination on the card and electronically produces the appropriate bit combinations in storage. Such translations (shown in Figure 10-7) are automatic; therefore, you need not be concerned with how the computer knows which bits to turn on and off.

When programming in RPG II, however, you do have to be concerned with zones and digits as they are represented inside the computer. The division of the card column into zone and digit portions is only for convenience.

On the Input sheet, you can specify record identification codes. If you choose to use only the zone portion of a character, you will be using the zone positions as they are in storage. Assume that a record identification code with

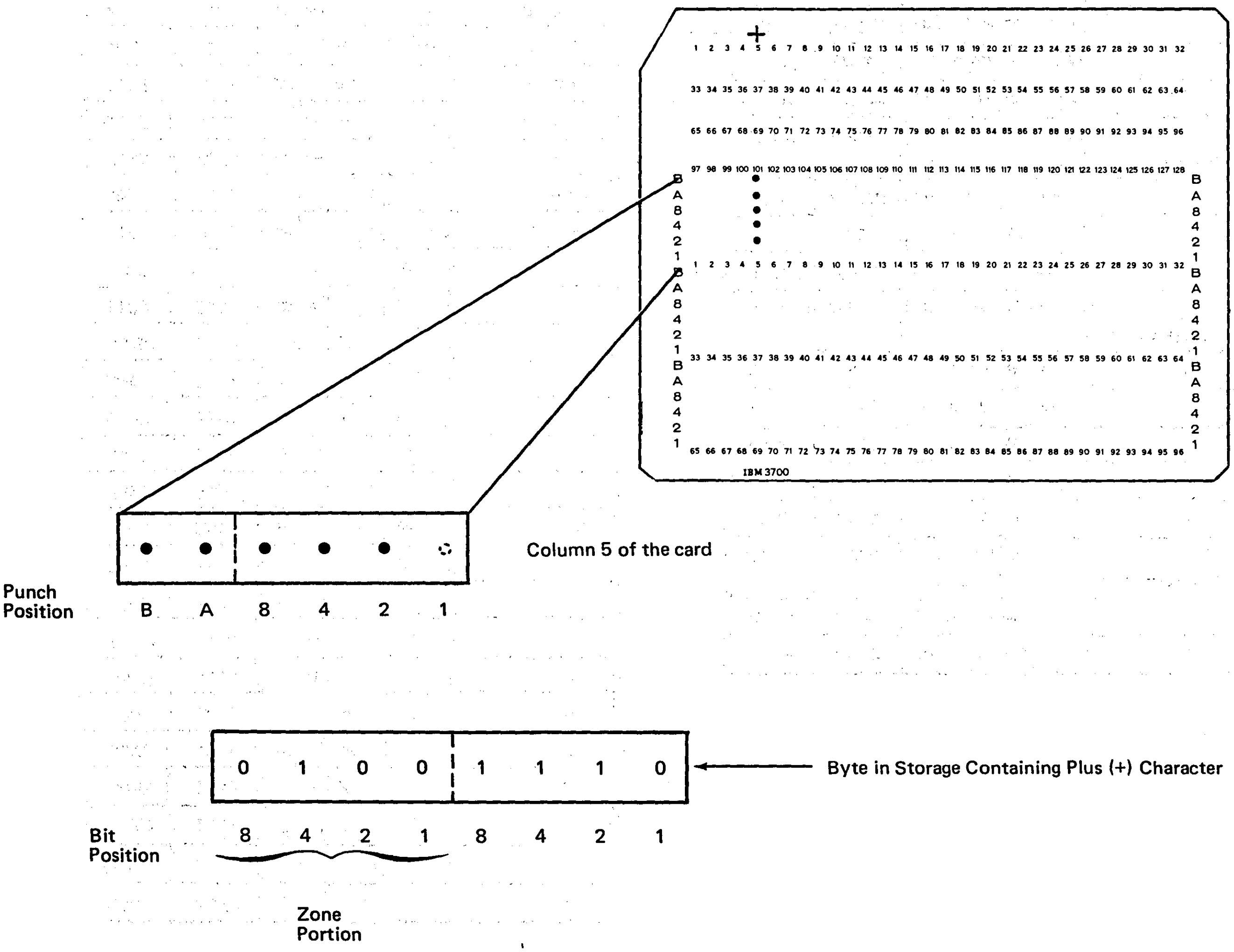


Figure 10-6. Similarity in Digit Portion of Byte and Card Column

Character	Punch Combination						Bit Combination	
	Zone		Digit					
	B	A	8	4	2	1	Zone	Digit
b̸ (blank)							0100	0000
¢	●	●	●		●		0100	1010
. (period)	●	●	●		●	●	0100	1011
<	●	●	●	●			0100	1100
(	●	●	●	●		●	0100	1101
+	●	●	●	●	●		0100	1110
|	●	●	●	●	●	●	0100	1111
&		●	●		●		0101	0000
!	●		●		●		0101	1010
$	●		●		●	●	0101	1011
*	●		●	●			0101	1100
)	●		●	●		●	0101	1101
;	●		●	●	●		0101	1110
¬	●		●	●	●	●	0101	1111
- (minus)	●						0110	0000
/		●				●	0110	0001
,		●	●		●	●	0110	1011
%		●	●	●			0110	1100
__ (underscore)		●	●	●		●	0110	1101
>		●	●	●	●		0110	1110
?		●	●	●	●	●	0110	1111
:			●		●		0111	1010
#			●		●	●	0111	1011
@			●	●			0111	1100
' (apostrophe)			●	●		●	0111	1101
=			●	●	●		0111	1110
"			●	●	●	●	0111	1111

Character	Punch Combination						Bit Combination	
	Zone		Digit					
	B	A	8	4	2	1	Zone	Digit
A	●	●				●	1100	0001
B	●	●			●		1100	0010
C	●	●			●	●	1100	0011
D	●	●		●			1100	0100
E	●	●		●		●	1100	0101
F	●	●		●	●		1100	0110
G	●	●		●	●	●	1100	0111
H	●	●	●				1100	1000
I	●	●	●			●	1100	1001
} or -0	●	●					1101	0000
J or -1	●					●	1101	0001
K or -2	●				●		1101	0010
L or -3	●				●	●	1101	0011
M or -4	●			●			1101	0100
N or -5	●			●		●	1101	0101
O or -6	●			●	●		1101	0110
P or -7	●			●	●	●	1101	0111
Q or -8	●		●				1101	1000
R or -9	●		●			●	1101	1001
S		●			●		1110	0010
T		●			●	●	1110	0011
U		●		●			1110	0100
V		●		●		●	1110	0101
W		●		●	●		1110	0110
X		●		●	●	●	1110	0111
Y		●	●				1110	1000
Z		●	●			●	1110	1001
+0		●					1111	0000
1						●	1111	0001
2					●		1111	0010
3					●	●	1111	0011
4				●			1111	0100
5				●		●	1111	0101
6				●	●		1111	0110
7				●	●	●	1111	0111
8			●				1111	1000
9			●			●	1111	1001

Figure 10-7. Bit and Punch Combinations for Characters

the zone of a $ character in column *1* is to turn on resulting indicator 21. If an input record is read with the character *J* in column *1*, indicator 21 will not turn on. Even though the card zone punches for $ and *J* are the same (both have the *B* zone punched), the bit combinations in storage for the $ and *J* do not have identical zone portions (Figure 10-8).

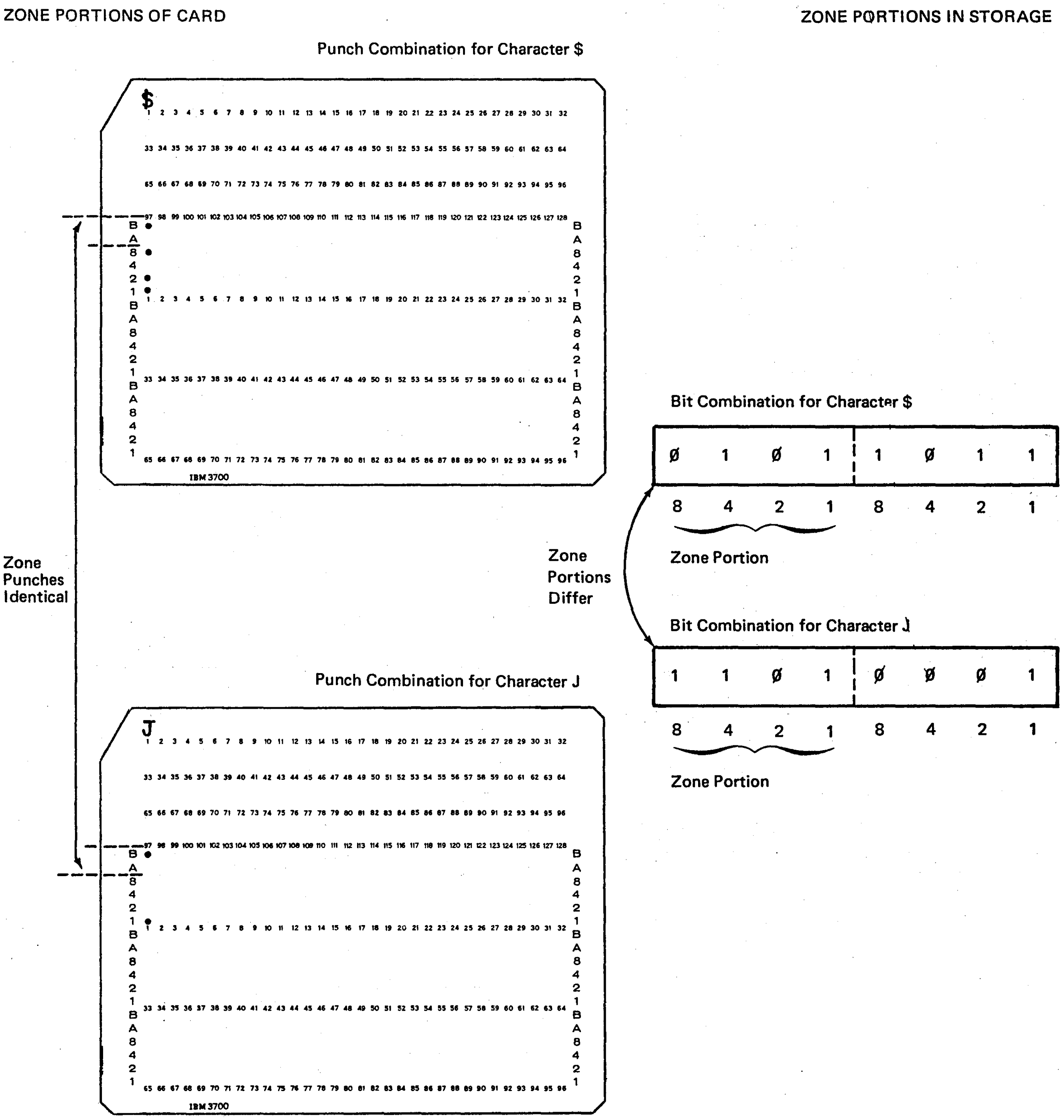


Figure 10-8. Difference in Zone Portions of a Byte and a Card Column

There are exceptions which should be noted in specifying that the zones of characters be used to identify records. According to Figure 10-9, the zone of the letter J is used to identify record type 01, while the zone of a minus sign is used to identify record type 02. Recorded on cards, both characters have a B zone punch. However, inside the computer, their zone representations differ.

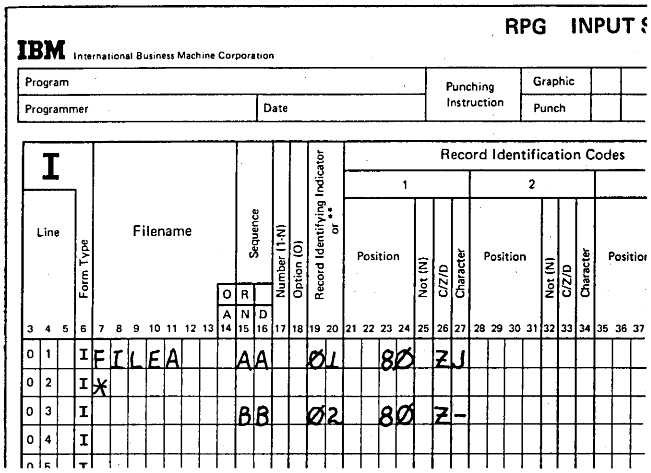
– = $\boxed{0110}$ 0000
8421

J = $\boxed{1101}$ 0001
8421

Although the zones differ, the RPG II program considers the two the same. Thus, a card with a minus punched might turn on either the 01 or 02 indicator. Likewise, a card with the letter J could turn on either indicator.

Similarly, the zone of the character *blank* is treated the same as the zone of 0 through 9. Also, the zone of & (ampersand) is treated the same as the zones of the letters A through I. To avoid confusion, you should not specify both (zones of the characters J and –; blank and 0-9; & and A-I) for identification of record types which are to be used in the same program.

Figure 10-10 shows the groups of characters that have zones that test as equal for purposes of record identification and the TESTZ operation. Notice that these groupings are different from the groupings of characters for purposes of collating sequence by zone (Figure 10-28).



IBM International Business Machine Corporation

RPG INPUT S

Program		Punching Instruction	Graphic		
Programmer	Date		Punch		

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Positio
01	I	FILEA	AA			Ø1	8Ø		Z	J					
02	I	*													
03	I		BB			Ø2	8Ø		Z	-					
04	I														
05	I														

Figure 10-9. Exception: Zone Representation Considered the Same

Character	Bit Combination		Collating Sequence of Zones
	Zone	Digit	
¢	0100	1010	1
. (period)	0100	1011	
<	0100	1100	
(	0100	1101	
+	0100	1110	
|	0100	1111	
!	0101	1010	2
$	0101	1011	
*	0101	1100	
)	0101	1101	
;	0101	1110	
¬	0101	1111	
/	0110	0001	3
,	0110	1011	
%	0110	1100	
_ (underscore)	0110	1101	
>	0110	1110	
?	0110	1111	
:	0111	1010	4
#	0111	1011	
@	0111	1100	
' (apostrophe)	0111	1101	
=	0111	1110	
"	0111	1111	
&	0101	0000	5
A	1100	0001	
B	1100	0010	
C	1100	0011	
D	1100	0100	
E	1100	0101	
F	1100	0110	
G	1100	0111	
H	1100	1000	
I	1100	1001	

Character	Bit Combination		Collating Sequence of Zones
	Zone	Digit	
(minus)	0110	0000	6
}	1101	0000	
J or -1	1101	0001	
K or -2	1101	0010	
L or -3	1101	0011	
M or -4	1101	0100	
N or -5	1101	0101	
O or -6	1101	0110	
P or -7	1101	0111	
Q or -8	1101	1000	
R or -9	1101	1001	
S	1110	0010	7
T	1110	0011	
U	1110	0100	
V	1110	0101	
W	1110	0110	
X	1110	0111	
Y	1110	1000	
Z	1110	1001	
b (blank)	0100	0000	8
+0	1111	0000	
1	1111	0001	
2	1111	0010	
3	1111	0011	
4	1111	0100	
5	1111	0101	
6	1111	0110	
7	1111	0111	
8	1111	1000	
9	1111	1001	

Figure 10-10. Character Groups With Zones that Test as Equal for Record ID and TESTZ

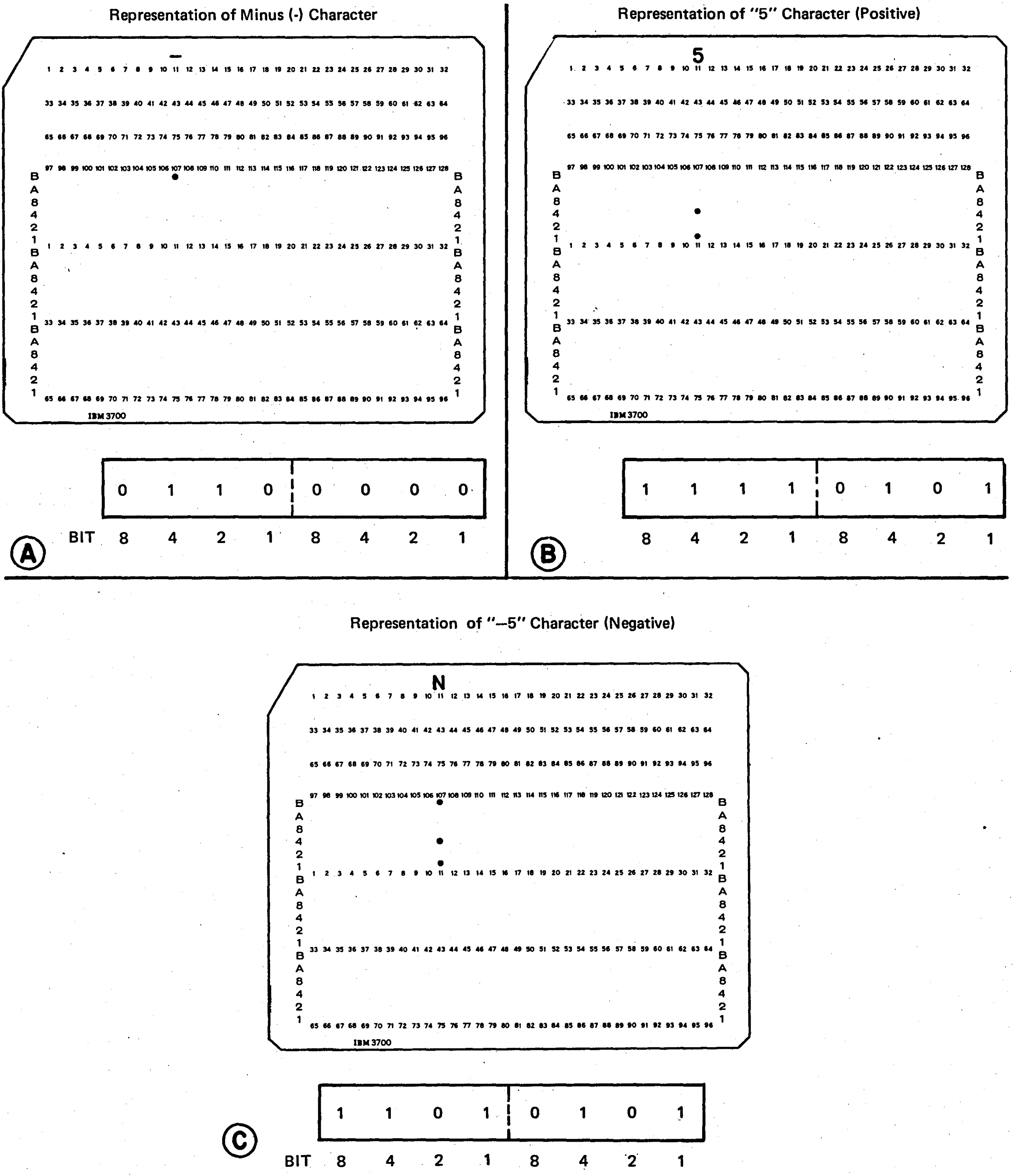
Consider another example which points out the difference in how negative numbers are stored and how you may think they are stored. The minus sign alone is represented on a card and in storage as shown in Figure 10-11, insert A. Only the zone portion of the card contains a punch. Figure 10-11, insert B, shows how a positive 5 is represented on a card and in storage. In this case, only the digit portion of the card contains punches. Note Figure 10-11, insert C, for the punch and bit combinations which represent a -5.

When checking the cards you can see that the digit punches for the positive 5 and the negative 5 are the same. Furthermore, the digit bits in storage for the two characters are also the same. The zone punch for -5 is the same as the zone punch for the minus sign character. However, the zone bits in storage for the two characters are not the same. Therefore, you should not always assume that, in storage, the zone bits for a negative number would be identical to the zone bits for the minus sign (Figure 10-11).

The reason is that the computer checks the *entire* punch combination (both zone and digit portions) of a column to determine which *zone* bits are to be on or off. Since the entire punch combinations (not only zone punches) for the minus character and the negative 5 are different, their zone bits in storage are also different.

A conclusion can be drawn from the previous examples: each unique punch combination is associated with a different bit combination. Of course, in discussing negative numbers before, we stated that the punch combinations of *-1* through *-9* and *-0* are the same as the punches for the letters *J* through *R* and } . Thus, the bit combinations for the negative numbers are also the same as those for *J* through *R*.

Consider again the number of positions available to represent a character. A characteristic of codes involving different combinations (such as bits on and off or punches) is that the greater the number of positions available to represent any one combination, the greater the number of combinations that are possible. As mentioned, with six punch positions, 64 unique punch combinations can be made, and, therefore, 64 different characters can be represented on a card. With eight bits (positions), 256 unique combinations of on-off bits can be created and, therefore, 256 different characters could be represented inside the computer. However, you only need 64 characters to program the computer; therefore, only 64 of its 256 bit combinations are associated with a printable character.



Representation of Minus (-) Character

BIT	8	4	2	1	8	4	2	1
	0	1	1	0	0	0	0	0

Representation of "5" Character (Positive)

	8	4	2	1	8	4	2	1
	1	1	1	1	0	1	0	1

Representation of "–5" Character (Negative)

BIT	8	4	2	1	8	4	2	1
	1	1	0	1	0	1	0	1

Figure 10-11. Representation of a Negative Number

Identifying Bit Combinations with Numerical Values

Each unique combination of eight bits can be associated with a numerical value. Before discussing how the numerical value is determined for a character, perhaps first you would like to know why numerical values are assigned.

As mentioned before, data can be represented on punched cards. Actually, after reading a card, the computer does not immediately determine what character is punched. It can, however, distinguish one punch combination from another punch combination. Furthermore, the particular combination of punches indicates to the computer which bits should be set on and off to represent that punch combination inside the machine. At this point, the representation on the card is just a particular group of punches and the representation in storage is merely a particular combination of on and off bits.

To use the byte of data for output, the computer must know what character to punch or print. This is done by associating a numerical value with each unique bit combination. The computer automatically knows that a certain value is related to a particular character, such as the value 209 indicates the character *J*.

Consider how a numerical value and how the character are determined. Each of the eight bits in a byte are assigned a number. The values begin with *1* for the *1* bit and are doubled for each of the next bits (Figure 10-12). By adding only the numbers which correspond to bits which are on *(1)*, a numerical value is obtained for a byte. As Figure 10-12 shows, first the punch combination (for the character *F*) in column 7 is translated into the bit combination in storage. The bits *on* result in a numerical value of 198, which the computer associates with the character *F*.

Any difference in the bit combination results in a difference in numerical value. Therefore, every character is associated with a different numerical value. The greatest numerical value which can be associated with a bit combination is 255 (all eight bits on), while the lowest numerical value is 0 (all eight bits off). This results in a total of 256 possible numerical values. Only 64 different characters can be represented on a 96 column card; therefore, we are concerned with only 64 of the different numerical values. However, as Figure 10-13 shows, the 64 numerical values associated with the characters can range anywhere from 0 through 255. The numerical values missing from the chart are not related to any printable character.

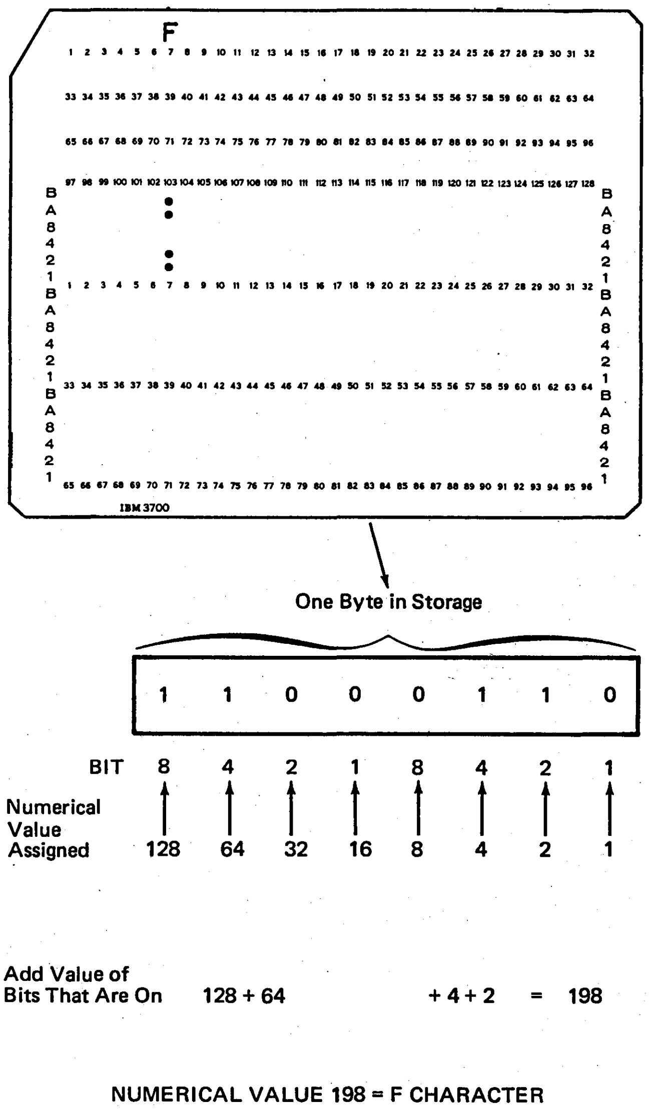

F
B A 8 4 2 1
IBM 3700
One Byte in Storage
1 1 0 0 0 1 1 0
BIT 8 4 2 1 8 4 2 1
Numerical Value Assigned 128 64 32 16 8 4 2 1
Add Value of Bits That Are On 128 + 64 + 4 + 2 = 198
NUMERICAL VALUE 198 = F CHARACTER

Figure 10-12. Determining a Numerical Value for a Character

Bit Combination	Numerical Value	Character
00000000	0	
00000001	1	
00000010	2	
00000011	3	
00000100	4	
00000101	5	
00000110	6	
00000111	7	
00001000	8	
00001001	9	
00001010	10	
00001011	11	
00001100	12	
00001101	13	
00001110	14	
00001111	15	
00010000	16	
00010001	17	
00010010	18	
00010011	19	
00010100	20	
00010101	21	
00010110	22	
00010111	23	
00011000	24	
00011001	25	
00011010	26	
00011011	27	
00011100	28	
00011101	29	
00011110	30	
00011111	31	
00100000	32	
00100001	33	
00100010	34	
00100011	35	
00100100	36	
00100101	37	
00100110	38	
00100111	39	
00101000	40	
00101001	41	
00101010	42	
00101011	43	
00101100	44	
00101101	45	
00101110	46	
00101111	47	
00110000	48	
00110001	49	
00110010	50	
00110011	51	
00110100	52	
00110101	53	
00110110	54	
00110111	55	
00111000	56	
00111001	57	
00111010	58	
00111011	59	
00111100	60	
00111101	61	
00111110	62	
00111111	63	
01000000	64	Blank
01000001	65	
01000010	66	
01000011	67	
01000100	68	
01000101	69	
01000110	70	
01000111	71	
01001000	72	
01001001	72	
01001010	74	¢
01001011	75	.
01001100	76	<
01001101	77	(
01001110	78	+
01001111	79	|
01010000	80	&
01010001	81	
01010010	82	
01010011	83	
01010100	84	
01010101	85	
01010110	86	
01010111	87	
01011000	88	
01011001	89	
01011010	90	!
01011011	91	$
01011100	92	*
01011101	93	)
01011110	94	;
01011111	95	¬
01100000	96	-
01100001	97	/
01100010	98	
01100011	99	
01100100	100	
01100101	101	

Figure 10-13 (Part 1 of 3). Numerical Values Associated with Characters

Bit Combination	Numerical Value	Character
01100110	102	
01100111	103	
01101000	104	
01101001	105	
01101010	106	
01101011	107	,
01101100	108	%
01101101	109	—
01101110	110	>
01101111	111	?
01110000	112	
01110001	113	
01110010	114	
01110011	115	
01110100	116	
01110101	117	
01110110	118	
01110111	119	
01111000	120	
01111001	121	
01111010	122	:
01111011	123	#
01111100	124	@
01111101	125	'
01111110	126	=
01111111	127	"
10000000	128	
10000001	129	
10000010	130	
10000011	131	
10000100	132	
10000101	133	
10000110	134	
10000111	135	
10001000	136	
10001001	137	
10001010	138	
10001011	139	
10001100	140	
10001101	141	
10001110	142	
10001111	143	
10010000	144	
10010001	145	
10010010	146	
10010011	147	
10010100	148	
10010101	149	
10010110	150	
10010111	151	
10011000	152	
10011001	153	
10011010	154	
10011011	155	
10011100	156	
10011101	157	
10011110	158	
10011111	159	
10100000	160	
10100001	161	
10100010	162	
10100011	163	
10100100	164	
10100101	165	
10100110	166	
10100111	167	
10101000	168	
10101001	169	
10101010	170	
10101011	171	
10101100	172	
10101101	173	
10101110	174	
10101111	175	
10110000	176	
10110001	177	
10110010	178	
10110011	179	
10110100	180	
10110101	181	
10110110	182	
10110111	183	
10111000	184	
10111001	185	
10111010	186	
10111011	187	
10111100	188	
10111101	189	
10111110	190	
10111111	191	
11000000	192	
11000001	193	A
11000010	194	B
11000011	195	C
11000100	196	D
11000101	197	E
11000110	198	F
11000111	199	G
11001000	200	H
11001001	201	I
11001010	202	
11001011	203	

Figure 10-13 (Part 2 of 3). Numerical Values Associated with Characters

Bit Combination	Numerical Value	Character
11001100	204	
11001101	205	
11001110	206	
11001111	207	
11010000	208	} or -0
11010001	209	J or -1
11010010	210	K or -2
11010011	211	L or -3
11010100	212	M or -4
11010101	213	N or -5
11010110	214	O or -6
11010111	215	P or -7
11011000	216	Q or -8
11011001	217	R or -9
11011010	218	
11011011	219	
11011100	220	
11011101	221	
11011110	222	
11011111	223	
11100000	224	
11100001	225	
11100010	226	S
11100011	227	T
11100100	228	U
11100101	229	V
11100110	230	W
11100111	231	X
11101000	232	Y
11101001	233	Z
11101010	234	
11101011	235	
11101100	236	
11101101	237	
11101110	238	
11101111	239	
11110000	240	0
11110001	241	1
11110010	242	2
11110011	243	3
11110100	244	4
11110101	245	5
11110110	246	6
11110111	247	7
11111000	248	8
11111001	249	9
11111010	250	
11111011	251	
11111100	252	
11111101	253	
11111110	254	
11111111	255	

Figure 10-13 (Part 3 of 3). Numerical Values Associated with Characters

Assigning Numerical Values to Zone and Digit Portions

You have seen how a single numerical value is determined for a combination of eight bits. The numerical value of a character in storage can also be expressed as a pair of numbers, rather than a single value. One number designates the value of only the four zone bits; the other number represents the value of the four digit bits.

You may be wondering why a character would ever be associated with two paired numbers, since it can be associated with just a single number. In certain jobs, you may be concerned with only the digit portion or only the zone portion of a character. For example, if records within a group are to be sequence checked only on the basis of the zone of a character, the computer must look at only the zone bits and determine a numerical value for the zone portion alone in order to make the comparison. Also, if you want to alter the collating sequence or translate a file, both to be discussed later, you must know the separate values for the zone and digit portions of a character.

Determining separate zone and digit values is similar to determining a single value for an entire bit combination; that is, values are assigned to each of the bit positions. The values which correspond to *on* bits *(1)* are then added to obtain a value.

To determine separate values, the zone and digit portions are each treated as separate 4-bit combinations. The four bits in each portion are assigned the values *1, 2, 4,* and *8* (Figure 10-14). The rightmost zone and digit bits each have the value *1*; while the leftmost bits in each portion are assigned the value *8.* A value for the zone portion of a byte is determined by adding only the values corresponding to zone bits which are on *(1).* Likewise, a digit value is obtained by considering only digit bits which are on.

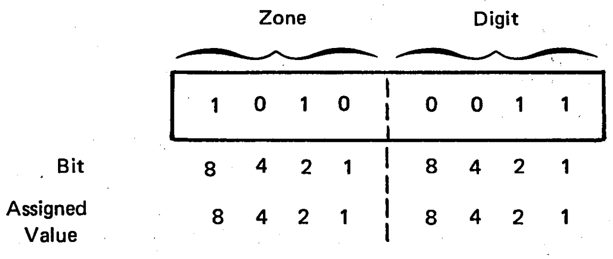


Figure 10-14. Assigning Values for Zone and Digit Portions of a Character

As Figure 10-15, insert A, shows, the bit combination for the slash (/) character produces a zone value of 6 and a digit value of 1. Putting the two values together, the entire character can be expressed as the value 61. Note, however, that this is not the same as the numerical value 61 in our decimal numbering system. If we were to determine a numerical value for the *entire* 8-bit combination, we would obtain the value 97 (Figure 10-15, insert B).

As mentioned before, with eight bits or positions in a byte, 256 different 8-bit combinations can be formed. The 256 combinations can be associated with the numerical values 0-255 (Figure 10-16, insert A). If either the zone or digit portion are considered separately, however, only four bits or positions are available. Therefore, a maximum of 16 different 4-bit combinations can be represented in either the zone or digit portion of a byte. The 16 zone or digit combinations can be associated with the values 0 through 15 (Figure 10-16, insert B).

The value obtained for a zone or digit bit combination is referred to as *hexadecimal* number. Hex means 6, while decimal referes to 10. Hexadecimal, then, means 6 + 10, or 16. A hexadecimal number can be any one of 16 possible values (0-15). Putting the two hexadecimal numbers for the zone and digit portions together gives a hexadecimal value for the entire character. *61* is the hexadecimal value for the / character. Keep in mind that this hexadecimal value is actually two separate values, one for the zone and one for the digit portion.

All of the 256 possible 8-bit combinations can be represented by a hexadecimal value; that is, two hexadecimal numbers. However, each hexadecimal number can take up only one position. If a zone portion has the value 15 and a digit portion has the value 12, the hexadecimal value for the character cannot be expressed as 1512. Consequently, a zone or digit portion whose numerical value is 10 or greater (2-position number) must be represented in a slightly different form. This is done by assigning a single letter as a substitute for the number. The letters *A* through *F* serve as the hexadecimal forms of the values 10 through 15 as shown in Figure 10-17.

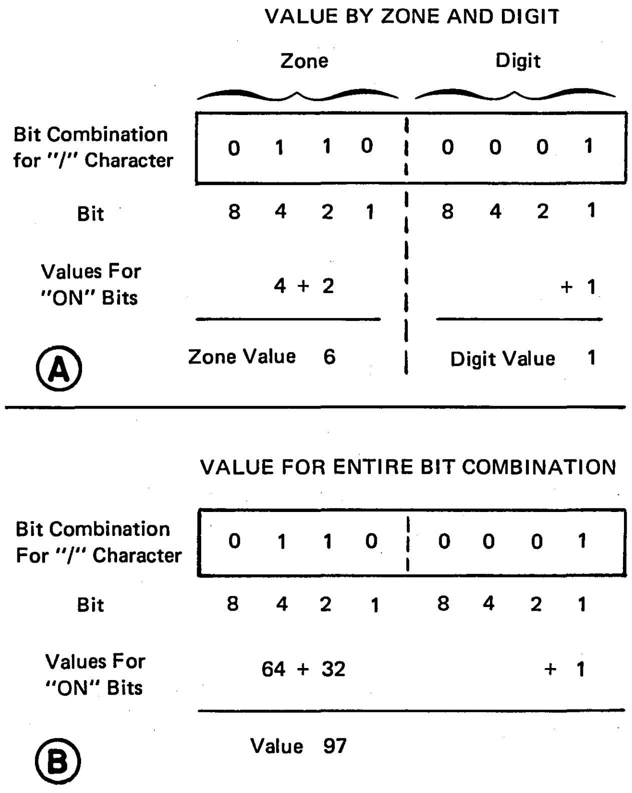


Figure 10-15. Difference in Value of Entire Character and Value of Zone and Digit Portions of Character

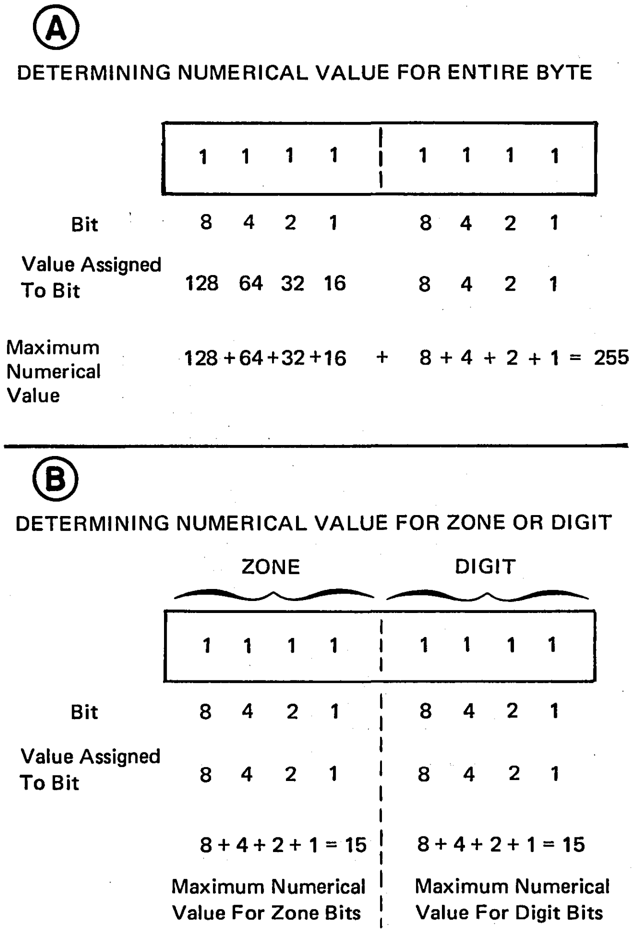


Figure 10-16. Maximum Values for Entire Character and for Zone and Digit Portions

Decimal Number	Hexadecimal Value
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Figure 10-17. Hexadecimal Values of Numbers 0-15

An 8-bit combination with a zone value of 15 and a digit value of 12 is expressed as having a hexadecimal value of FC. Because the complete numbering series is composed of numbers *0* through *9* followed by letters *A* through *F,* a hexadecimal value for the zone and digit portion of an 8-bit combination can appear as a pair of numbers (61), a letter and a number (C4, 4F), or a pair of letters (DB).

Since zone and digit values are determined separately, a single combination of eight bits can have the same hexadecimal number for both the zone and digit portion. Thus, 11, 22, 33, AA, and other such values represent 8-bit combinations which have the same bits on in both their zone and digit portions.

Entirely different 8-bit combinations can have identical zone hexadecimal numbers *or* identical digit hexadecimal numbers, but not both. That is, the zone portion of one character can contain the same bits on and off as the zone portion of another character. In such a case, the identical zone bit combinations would give identical zone hexadecimal values. However, if they are different characters and the zone values are identical, the digit bits and, thus, the digit values, must differ. This is because no two characters can have the same 8-bit combination.

Figure 10-18 shows the hexadecimal values associated with the 64 printable characters the computer recognizes. The hexadecimal values are in sequence just as the regular numerical values are. Furthermore, the hexadecimal value associated with a character is equivalent to the numerical value associated with that character. However, there is no need for you to be able to translate back and forth between numerical values and hexadecimal values. If you must use a character's hexadecimal vlaue in your programming, you can refer to the chart showing the appropriate value.

Character	Hexadecimal Value	Numerical Value
Blank	40	64
¢	4A	74
.	4B	75
<	4C	76
(	4D	77
+	4E	78
|	4F	79
&	50	80
!	5A	90
$	5B	91
*	5C	92
)	5D	93
;	5E	94
¬	5F	95
-	60	96
/	61	97
,	6B	107
%	6C	108
_	6D	109
>	6E	110
?	6F	111
:	7A	122
#	7B	123
@	7C	124
'	7D	125
=	7E	126
"	7F	127
A	C1	193
B	C2	194
C	C3	195
D	C4	196
E	C5	197
F	C6	198
G	C7	199
H	C8	200
I	C9	201
} or -0	D0	208
J or -1	D1	209
K or -2	D2	210
L or -3	D3	211
M or -4	D4	212
N or -5	D5	213
O or -6	D6	214
P or -7	D7	215
Q or -8	D8	216
R or -9	D9	217
S	E2	226
T	E3	227
U	E4	228
V	E5	229
W	E6	230
X	E7	231
Y	E8	232
Z	E9	233
0	F0	240
1	F1	241
2	F2	242
3	F3	243
4	F4	244
5	F5	245
6	F6	246
7	F7	247
8	F8	248
9	F9	249

Figure 10-18. Hexadecimal Values Associated with Characters

Saving Disk Storage Space

As you have learned, each byte of storage, whether on disk or in the computer, can contain one character. That character can be a decimal number or an alphabetic or special character. The format of the characters is known as *unpacked decimal format.* Each byte of storage is divided into a 4-bit zone and a 4-bit digit part. Figure 10-19 shows the unpacked decimal format.

The zone part of the low-order (rightmost) byte indicates whether the decimal number is positive or negative. In unpacked decimal format, the zone part is included for each digit in a decimal number; however, only the zone over the low-order digit serves as the sign. The low-order digit is the only digit which makes use of the zone portion. Figure 10-20 shows the unpacked decimal format for decimal number 9,269.

Packed Decimal Format

In *packed decimal format* means that one byte of storage can contain two decimal numbers. A decimal number will occupy the zone portion which is unused in unpacked decimal format. This format allows you to put almost twice as much data into a byte as you can using the unpacked decimal format.

The low-order byte in packed decimal format is also divided into two 4-bit parts. Each byte, except the low-order byte, contains one decimal digit in each 4-bit part. The low-order byte contains a decimal digit in the leftmost 4-bit part (bits 0-3) and the sign of the decimal field in the rightmost 4-bit part (bits 4-7). Figure 10-21 shows packed decimal format.

The sign part of the low-order byte is used to indicate whether the numeric value represented in the digit parts is positive or negative. Compare how the decimal number 9,269 is represented in packed decimal format (Figure 10-22) with its unpacked representation (Figure 10-20).

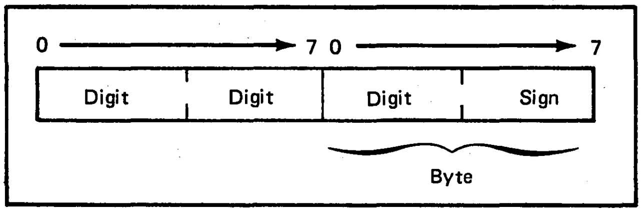


Figure 10-21. Packed Decimal Format

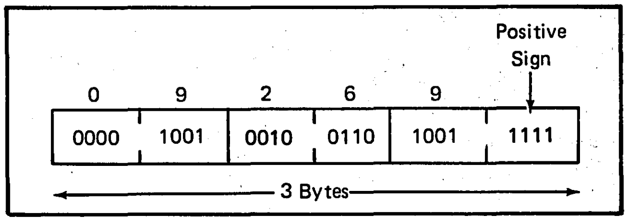


Figure 10-22. Packed Format of Decimal Number 9,269

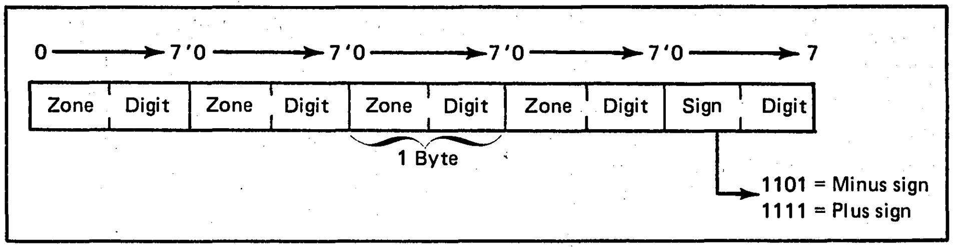


Figure 10-19. Unpacked Decimal Format

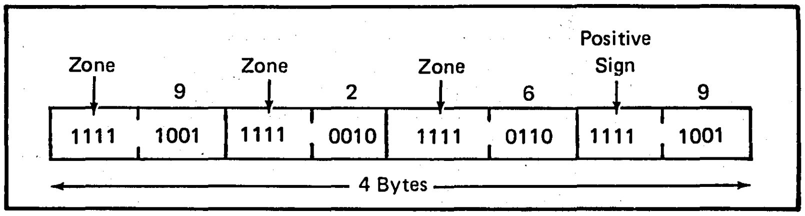


Figure 10-20. Unpacked Format of Decimal Number 9,269

You can specify packed input, output, table, or array fields:

- Packed input fields. Enter a *P* in column 43 of the Input sheet. This causes the data to be unpacked before it is stored.

- Packed output fields. Enter a *P* in column 44 of the Output-Format sheet. This causes the data to be packed before it is written out.

- Packed table or array fields. Enter a *P* in column 43 and/or 55 of the Extension sheet. The data will be unpacked before it is stored. Packed tables or arrays are allowed only at pre-execution time.

Since data must be represented in unpacked decimal format once it is inside the computer, you must give the RPG program an indication when input fields are in a different format.

Because data must be represented in unpacked decimal format before it can be processed, unpacked decimal fields may be stored on disk to eliminate converting the fields from packed to unpacked format during input. However, storing unpacked fields on disk requires more space than storing packed fields.

Binary Format

You can save even more disk space than in packed decimal format by storing numeric data in *binary format.* In binary format, each numeric field on disk must be either two or four bytes long. Each two-byte binary field can contain a value equivalent to four decimal places; each four-byte binary field can contain a value equivalent to nine decimal places. In other words, a numeric value in binary format occupies approximately half as many bytes of disk storage as the equivalent value in unpacked decimal format.

Each two-byte binary field consists of a sign bit followed by a 15-bit numeric value. This value can be as large as 9,999. When a two-byte binary field from disk storage is read into the computer, the RPG II program converts it to a four-byte unpacked decimal field. Figure 10-23 shows a two-byte field in binary format.

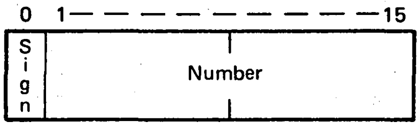


Figure 10-23. Two-Byte Field in Binary Format

Each four byte binary field consists of a sign bit followed by a 31-bit numeric value. This value can be as large as 999,999,999. When a four-byte binary field from disk storage is read into the computer, the RPG II program converts it to a nine-byte unpacked decimal field. A four-byte binary field is shown in Figure 10-24.

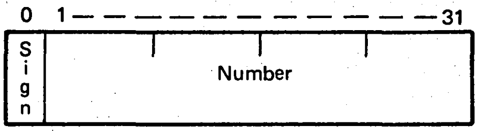


Figure 10-24. Four-Byte Field in Binary Format

In each case, the sign portion of the high-order byte (leftmost) is used to indicate whether the numeric value is positive or negative. Notice that in the binary format the zone portion of the decimal number is not given. Compare how the decimal number 9,269 is represented in binary format (Figure 10-25) with its packed and unpacked representation (Figure 10-20 and 10-22).

Positive Sign	0	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	= 9,269*
0	0	1	0	0	1	0	0	0	0	1	1	0	1	0	1	

* The numeric value for each binary byte is obtained by adding the numbers which correspond to the bits that are on. (Bits that are on are represented as 1's.) The sign bit is not included in the value of the number. The bit to the right of the sign bit is always 0, because the maximum value of a two-byte binary field is 9,999.

Figure 10-25. Binary Format of Decimal Number 9,269

Since data must be represented in unpacked decimal format when it is inside the computer, you must indicate to the RPG II program when fields are in another format. You can specify binary input, output, table, or array fields:

- Binary input fields. Enter a *B* in column 43 of the Input sheet. The data is then converted into decimal before it is stored.

- Binary output fields. Enter a *B* in column 44 of the Output-Format sheet. The data is then converted into binary before it is stored.

- Binary table or array fields. Enter a *B* in column 43 and/or 55 of the Extension sheet. The data will be converted to decimal before it is stored. Binary tables or arrays are allowed only at pre-execution time.

COLLATING SEQUENCE OF CHARACTERS

To perform data processing applications efficiently, you usually organize your information in some order or sequence. Imagine trying to locate a person's phone number if the names in a telephone book were not in alphabetical order. Of course, before using this order or sequence, you must know what it is. Through the learning process, you know that alphabetical order means that *A* comes before *B, B* before *C,* and so on. Likewise, in numerical sequence, *1* is less than *2, 2* less than *3*, and so on.

In most of your data processing applications, the computer must be able to recognize an order or sequence of data. For example, if you instruct the computer to sequence check a file according to an alphabetic department code on each record, it must be able to determine if the department *T* record should appear before the department *X* record, or vice versa. In another instruction, perhaps the computer is to compare two quantities, such as *3* and *8,* and turn on an indicator if the first quantity is less than the second quantity. The computer must determine if *3* is less than *8* or if *8* is less than *3.*

The previous two tasks would be easy for you to perform because, through memorization or habit, you know the natural order of the alphabetic characters *A* through *Z* and the numbers *0* through *9.* However, a computer has not memorized such orders. For the computer to perform these tasks, an order or sequence of characters must be established.

To further point out the need for a set order of characters, assume the computer must check to make sure records in a file are in proper order according to a department field. Some department codes are alphabetic, as department *A;* while some department codes are numbers, such as department *8.* Should the numeric department records appear before the alphabetic department records, or vice versa? It is likely that the answer would depend on who is asked. However, for efficient data processing, you certainly do not want records sorted one way one time and another way the next time. Thus, the computer must use *one* set order of characters.

Every character recognized by the computer must hold a certain position in this order in relation to the position of the rest of the characters. Such an order is referred to as a *collating sequence* of characters. By definition, *to collate* means to arrange or verify that data appears in proper order or sequence.

There can be any number of collating sequences. The sequence used depends on the particular order in which characters are to be recognized. In any case, the computer should use only one collating sequence at a time.

The standard collating sequence of 64 characters is shown in Figure 10-26. The blank, which is the first character, is considered as the lowest in the sequence while the number *9,* the last character, is the highest in the sequence. Note that all of the special characters, except the { (brace), are first in this sequence, followed by the alphabetic characters *A* through *Z* in their natural order, and then the numbers *0* through *9* in their natural order. The only character which you might not expect to be in its position is the { which comes between the letters *I* and *J.*

This collating sequence is the order used by the computer for the purpose of sorting cards, comparing numbers to determine which is greater or less, checking the sequence of records in a file, and matching records from two files to determine which record should be processed next. According to the collating sequence, the computer compares two characters to determine if one comes before or after the other, or is less than or greater than the other. Of course, you specify which characters (or fields of characters) are to be compared.

For sorting, sequence checking, and matching, you can specify an ascending or descending collating sequence. For example, if records are to be in ascending sequence, the characters being checked should be in the order shown in Figure 10-26. That is, a card with a blank should come before a card with the letter *K.* (The blank character is lower in sequence than the letter *K.*) Likewise, a card with the letter *K* should come before any records containing one of the numbers *0* through *9.* If you specify descending sequence, the computer compares to make sure they are in the opposite order, the characters higher in sequence coming first.

1	Blank	23	#	45	Q
2	¢	24	@	46	R
3	.	25	'	47	S
4	<	26	=	48	T
5	(	27	"	49	U
6	+	28	A	50	V
7	|	29	B	51	W
8	&	30	C	52	X
9	!	31	D	53	Y
10	$	32	E	54	Z
11	*	33	F	55	0
12	)	34	G	56	1
13	;	35	H	57	2
14	¬	36	I	58	3
15	- (minus)	37	}	59	4
16	/	38	J	60	5
17	,	39	K	61	6
18	%	40	L	62	7
19	_(underscore)	41	M	63	8
20	>	42	N	64	9
21	?	43	O		
22	:	44	P		

Figure 10-26. Standard Collating Sequence

As mentioned, a computer cannot memorize the order of characters; it must use another method for *remembering* the collating sequence. To do this, it uses the values associated with characters to determine each character's relation to another character in the sequence.

In a previous discussion, a value is calculated for each bit combination in storage. The value can be thought of as a single numerical value for the entire 8-bit combination or as a 2-digit hexadecimal value, which is actually one hexadecimal number for each 4-bit combination (zone and digit). A hexadecimal value is another way of representing a numerical value.

Once a value is calculated, the computer uses it to determine which character is represented. Thus, the numerical value 193 (same as hexadecimal value C1) is associated with the character *A* while the numerical value 243 (hexadecimal value F3) is associated with the numeric character *3.* Perhaps you wonder why a particular value, such as 193 (C1) is related to the letter *A,* rather than a different value.

The values associated with the 64 characters were originally assigned such that the natural sequence of the values corresponds with the positions characters are to hold within the collating sequence. For example, the character *A* is associated with value 193 (hexadecimal C1), *B* with 194 (C2), *C* with 195 (C3), and so on. Just as *A* is lower than *B* and *B* is lower than *C* in the collating sequence, 193 (C1) is less than 194 (C2), and 194 (C2) is less than 195 (C3).

Figure 10-27 shows the 256 possible bit combinations, their numerical and hexadecimal values, and the characters associated with each. In this list of bit combinations, the numerical values are in order from 0 through 255 (hexadecimal values 00 through FF), and the associated characters are in the standard ascending collating sequence.

As you can readily see, the value associated with a character does not always *immediately* follow the value associated with the previous character in the sequence. For example, the character *S* follows the character *R* in the collating sequence of characters. However, the numerical value of *S*, 226 (hexadecimal E2), does not immediately follow the

Bit Combination	Character	Hexadecimal Value	Numerical Value
00000000		00	0
00000001		01	1
00000010		02	2
00000011		03	3
00000100		04	4
00000101		05	5
00000110		06	6
00000111		07	7
00001000		08	8
00001001		09	9
00001010		0A	10
00001011		0B	11
00001100		0C	12
00001101		0D	13
00001110		0E	14
00001111		0F	15
00010000		10	16
00010001		11	17
00010010		12	18
00010011		13	19
00010100		14	20
00010101		15	21
00010110		16	22
00010111		17	23
00011000		18	24
00011001		19	25
00011010		1A	26
00011011		1B	27
00011100		1C	28
00011101		1D	29
00011110		1E	30
00011111		1F	31
00100000		20	32
00100001		21	33
00100010		22	34
00100011		23	35
00100100		24	36
00100101		25	37
00100110		26	38
00100111		27	39
00101000		28	40
00101001		29	41
00101010		2A	42
00101011		2B	43
00101100		2C	44
00101101		2D	45
00101110		2E	46
00101111		2F	47
00110000		30	48
00110001		31	49
00110010		32	50
00110011		33	51
00110100		34	52
00110101		35	53
00110110		36	54
00110111		37	55
00111000		38	56
00111001		39	57
00111010		3A	58
00111011		3B	59
00111100		3C	60
00111101		3D	61
00111110		3E	62
00111111		3F	63
01000000	Blank	40	64
01000001		41	65
01000010		42	66
01000011		43	67
01000100		44	68
01000101		45	69
01000110		46	70
01000111		47	71
01001000		48	72
01001001		49	73
01001010	¢	4A	74
01001011	.	4B	75
0100110	<	4C	76
01001101	(	4D	77
01001110	+	4E	78
01001111	|	4F	79
01010000	&	50	80
01010001		51	81
01010010		52	82
01010011		53	83
01010100		54	84
01010101		55	85
01010110		56	86
01010111		57	87
01011000		58	88
01011001		59	89
01011010	!	5A	90
01011011	$	5B	91
01011100	*	5C	92
01011101	)	5D	93
01011110	;	5E	94
01011111	¬	5F	95
01100000	-	60	96
01100001	/	61	97
01100010		62	98
01100011		63	99
01100100		64	100
01100101		65	101

Figure 10-27 (Part 1 of 3). Characters and Values Associated with the 256 Bit Combinations

numerical value of *R*, 217 (hexadecimal D9). The reason for the gap is because the bit combinations with the numerical values 218 through 225 are not associated with any of the 64 printable characters. Regardless, the computer determines that *R* is lower in sequence than (comes before) *S* because the value of *R* (217 or hexadecimal D9) is less than the value of *S* (226 or hexadecimal E2).

Bit Combination	Character	Hexadecimal Value	Numerical Value
01100110		66	102
01100111		67	103
01101000		68	104
01101001		69	105
01101010		6A	106
01101011	,	6B	107
01101100	%	6C	108
01101101	—	6D	109
01101110	>	6E	110
01101111	?	6F	111
01110000		70	112
01110001		71	113
01110010		72	114
01110011		73	115
01110100		74	116
01110101		75	117
01110110		76	118
01110111		77	119
01111000		78	120
01111001		79	121
01111010	:	7A	122
01111011	#	7B	123
01111100	@	7C	124
01111101	'	7D	125
01111110	=	7E	126
01111111	"	7F	127
10000000		80	128
10000001		81	129
10000010		82	130
10000011		83	131
10000100		84	132
10000101		85	133
10000110		86	134
10000111		87	135
10001000		88	136
10001001		89	137
10001010		8A	138
10001011		8B	139
10001100		8C	140
10001101		8D	141
10001110		8E	142
10001111		8F	143
10010000		90	144
10010001		91	145
10010010		92	146
10010011		93	147
10010100		94	148
10010101		95	149
10010110		96	150
10010111		97	151
10011000		98	152

Bit Combination	Character	Hexadecimal Value	Numerical Value
10011001		99	153
10011010		9A	154
10011011		9B	155
10011100		9C	156
10011101		9D	157
10011110		9E	158
10011111		9F	159
10100000		A0	160
10100001		A1	161
10100010		A2	162
10100011		A3	163
10100100		A4	164
10100101		A5	165
10100110		A6	166
10100111		A7	167
10101000		A8	168
10101001		A9	169
10101010		AA	170
10101011		AB	171
10101100		AC	172
10101101		AD	173
10101110		AE	174
10101111		AF	175
10110000		B0	176
10110001		B1	177
10110010		B2	178
10110011		B3	179
10110100		B4	180
10110101		B5	181
10110110		B6	182
10110111		B7	183
10111000		B8	184
10111001		B9	185
10111010		BA	186
10111011		BB	187
10111100		BC	188
10111101		BD	189
10111110		BE	190
10111111		BF	191
11000000		C0	192
11000001	A	C1	193
11000010	B	C2	194
11000011	C	C3	195
11000100	D	C4	196
11000101	E	C5	197
11000110	F	C6	198
11000111	G	C7	199
11001000	H	C8	100
11001001	I	C9	101
11001010		CA	202
11001011		CB	203

Figure 10-27 (Part 2 of 3). Characters and Values Associated with the 256 Bit Combinations

Bit Combination	Character	Hexadecimal Value	Numerical Value
11001100		CC	204
11001101		CD	205
11001110		CE	206
11001111		CF	207
11010000	}	D0	208
11010001	J	D1	209
11010010	K	D2	210
11010011	L	D3	211
11010100	M	D4	212
11010101	N	D5	213
11010110	O	D6	214
11010111	P	D7	215
11011000	Q	D8	216
11011001	R	D9	217
11011010		DA	218
11011011		DB	219
11011100		DC	220
11011101		DD	221
11011110		DE	222
11011111		DF	223
11100000		E0	224
11100001		E1	225
11100010	S	E2	226
11100011	T	E3	227
11100100	U	E4	228
11100101	V	E5	229
11100110	W	E6	230
11100111	X	E7	231
11101000	Y	E8	232
11101001	Z	E9	233
11101010		EA	234
11101011		EB	235
11101100		EC	236
11101101		ED	237
11101110		EE	238
11101111		EF	239
11110000	0	F0	240
11110001	1	F1	241
11110010	2	F2	242
11110011	3	F3	243
11110100	4	F4	244
11110101	5	F5	245
11110110	6	F6	246
11110111	7	F7	247
11111000	8	F8	248
11111001	9	F9	249
11111010		FA	250
11111011		FB	251
11111100		FC	252
11111101		FD	253
11111110		FE	254
11111111		FF	255

Figure 10-27 (Part 3 of 3). Characters and Values Associated with the 256 Bit Combinations

Collating By Zone Or Digit

You learned from a previous discussion that the zone and digit portions of characters can be treated as separate and distinct groups of four bits, each with its own hexadecimal number.

Different characters may have identical zone bits or identical digit bits, but not both. Consequently, different characters may be associated with the same zone hexadecimal number of the same digit hexadecimal number but not both. As an example, the character *A* is associated with the value C1, *B* is associated with C2, and *K* is associated with D2. *A* has the same zone value (C) as *B,* while *K* has the same digit value (2) as *B.* However, *B* is the only character with both a zone value of C and a digit value of 2.

In most data processing tasks, the computer uses entire characters or the values of those characters to make comparisons, to determine which is greater or less, and so on. However, for certain purposes, such as sorting cards, you may wish to have the computer check only the zone or only the digit portion of characters. In such a case, the computer must use a collating sequence based on zone or digit values rather than the standard collating sequence based on the entire value.

If the computer uses a collating sequence based on the zone portions of characters, any differences in the digit bits are ignored. Only the value of the zone bits are considered. The reverse occurs if a collating sequence based on the digit portions of characters is to be used.

The fact that certain characters have the same zone or digit values can be used to group characters within a collating sequence. On the basis of zone values, the 64 printable characters are divided into eight groups (Figure 10-28). The zone bits (and values) are identical for all characters within a group. If collating is to be on the basis of digit values, the characters can be divided into 16 groups (Figure 10-29). In such a case, digit bits (and values) are identical for all characters within a particular group.

Using the standard collating sequence, the computer considers each character to hold a specific position in the sequence. Therefore, no two characters can be considered equal; one must come before another or be less than another character.

On the other hand, using a collating sequence based on zones or digits, one *group* of characters follows another *group* of characters. The characters within a group can occupy any position within that group. Thus, there is an order of groups but no particular order of characters within a group.

Character	Bit Combination		Collating Sequence of Zones
	Zone	Digit	
b (blank)	0100	0000	1
¢	0100	1010	
(period)	0100	1011	
<	0100	1100	
(	0100	1101	
+	0100	1110	
|	0100	1111	
&	0101	0000	2
!	0101	1010	
$	0101	1011	
*	0101	1100	
)	0101	1101	
;	0101	1110	
¬	0101	1111	
- (minus)	0110	0000	3
/	0110	0001	
,	0110	1011	
%	0110	1100	
_ (underscore)	0110	1101	
>	0110	1110	
?	0110	1111	
:	0111	1010	4
#	0111	1011	
@	0111	1100	
' (apostrophe)	0111	1101	
=	0111	1110	
"	0111	1111	
A	1100	0001	5
B	1100	0010	
C	1100	0011	
D	1100	0100	
E	1100	0101	
F	1100	0110	
G	1100	0111	
H	1100	1000	
I	1100	1001	
}	1101	0000	6
J or -1	1101	0001	
K or -2	1101	0010	
L or -3	1101	0011	
M or -4	1101	0100	
N or -5	1101	0101	
O or -6	1101	0110	
P or -7	1101	0111	
Q or -8	1101	1000	
R or -9	1101	1001	
S	1110	0010	7
T	1110	0011	
U	1110	0100	
V	1110	0101	
W	1110	0110	
X	1110	0111	
Y	1110	1000	
Z	1110	1001	
+0	1111	0000	8
1	1111	0001	
2	1111	0010	
3	1111	0011	
4	1111	0100	
5	1111	0101	
6	1111	0110	
7	1111	0111	
8	1111	1000	
9	1111	1001	

Figure 10-28. Collating Sequence by Zone

Character	Bit Combination		Collating Sequence of Digits
	Zone	Digit	
b̸ (blank)	0100	0000	1
&	0101	0000	
- (minus)	0110	0000	
}	1101	0000	
+0	1111	0000	
/	0110	0001	2
A	1100	0001	
J or -1	1101	0001	
1	1111	0001	
B	1100	0010	3
K or -2	1101	0010	
S	1110	0010	
2	1111	0010	
C	1100	0011	4
L or -3	1101	0011	
T	1110	0011	
3	1111	0011	
D	1100	0100	5
M or -4	1101	0100	
U	1110	0100	
4	1111	0100	
E	1100	0101	6
N or -5	1101	0101	
V	1110	0101	
5	1111	0101	
F	1100	0110	7
O or -6	1101	0110	
W	1110	0110	
6	1111	0110	
G	1100	0111	8
P or -7	1101	0111	
X	1110	0111	
7	1111	0111	
H	1100	1000	9
Q or -8	1101	1000	
Y	1110	1000	
8	1111	1000	
I	1100	1001	10
R or-9	1101	1001	
Z	1110	1001	
9	1111	1001	
¢	0100	1010	11
!	0101	1010	
:	0111	1010	
. (period)	0100	1011	12
$	0101	1011	
,	0110	1011	
#	0111	1011	
<	0100	1100	13
*	0101	1100	
%	0110	1100	
@	0111	1100	
(	0100	1101	14
)	0101	1101	
— (underscore)	0110	1101	
' (apostrophe)	0111	1101	
+	0100	1110	15
;	0101	1110	
>	0110	1110	
=	0111	1110	
|	0100	1111	16
¬	0101	1111	
?	0110	1111	
"	0111	1111	

Figure 10-29. Collating Sequence by Digit

Note that in the collating sequence by zone shown in Figure 10-28, any character in group *5* is considered lower in sequence than any character in group *6*. If records are sorted in ascending order by zone, a record with the letter D (group 5) comes before a record with the letter N (group 6).

Now consider a case in which characters from the same zone group are to be compared. Assume one record contains the letter *D* (group 5) and the next record contains the letter *F* (group 5). Which should be sorted first according to a collating sequence based on zones? Since the computer ignores the digit bits of each character, they are considered equal because they both have the same zone value. Therefore, no one character must come earlier in the sequence than another character *from the same group.* The sequence of the characters is the same order in which the records are read. Thus, if the *D* card is read first by a sort program, the *D* record comes before the *F* record. On the other hand, if the *F* card is read first, the *F* record comes before the *D* record. In either case, the records are in proper sequence based on zones.

ALTERING THE COLLATING SEQUENCE

A collating sequence is the order in which characters are arranged. As you know, all characters are associated with different numerical values in order that the computer may recognize them. The sequence of numerical values (ascending or descending sequence) determines the order in which characters associated with the values are recognized.

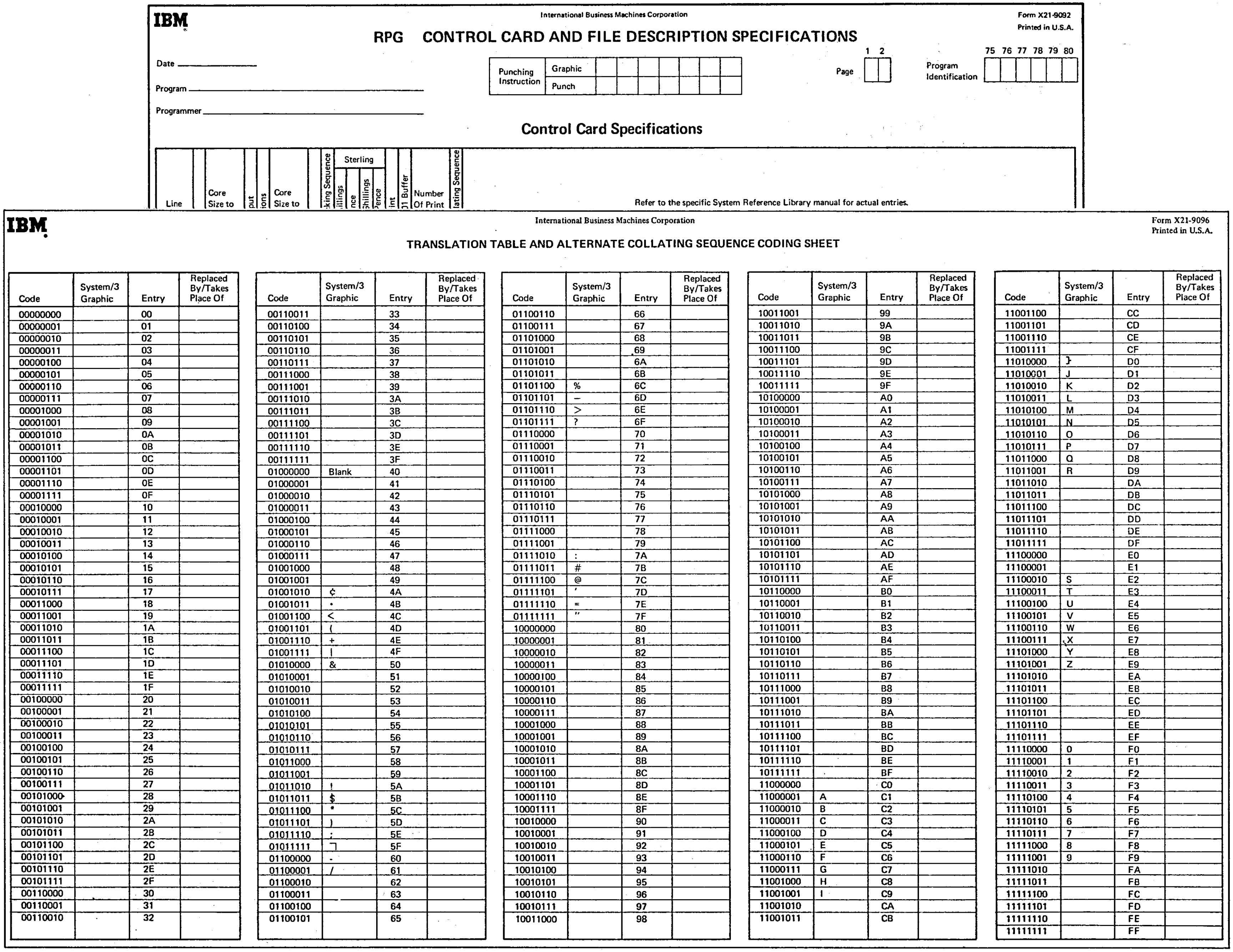
The association of a particular character with a numerical value is an arbitrary decision. Thus, the collating sequence itself is arbitrary. System/3 is programmed to expect the collating sequence discussed previously in the section *Collating Sequence of Characters.* This does not mean, however, that you must always use this sequence. You can change it and there may be times when you desire to do so. For example, you may want alphabetic characters to follow the numbers instead of preceding them. Suppose that a company originally started with a few departments. The departments were assigned numbers from 01-99. Two columns were devoted to department numbers in various records. The company expanded and departments increased. Soon there were more than 99 departments. To avoid having to change the department field from two to three characters in all records, the manager decided to use the letters of the alphabet to represent department numbers: 99, A0, A1, etc. In this case, *A* must follow the number *9* in the sequence. Thus it is necessary to alter the collating sequence so that numbers come before alphabetic characters.

There can be other reasons than the one just explained for altering the collating sequence. Your language may demand that you have characters such as *Ä, A′, Ö, Ë, E′* included in the alphabetic sequence (*A, Ä, B*). Since the 64 graphics do not include these characters, other seldom used characters can be substituted for them and repositioned in the collating sequence. For example, a number-symbol (#) repositioned between the letters *A* and *B* can substitute for an *Ä*; an at-symbol (@) repositioned between *O* and *P* substitutes for an *Ö.*

These are only a few reasons for altering the collating sequence. You may have others. Just remember that you can alter the collating sequence in any way that fits your needs.

Specifying Changes in Collating Sequence

To change the collating sequence, you must associate characters with different numerical values. The following sections will explain how this is done.



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic							
	Punch							

Page 1 2 □□

Program Identification 75 76 77 78 79 80 □□□□□□

Control Card Specifications

Line | Core Size to | put | ions | Core Size to | king Sequence | illings | nce | Sterling | hillings | ence | int | 1 Buffer | Number Of Print | ating Sequence

Refer to the specific System Reference Library manual for actual entries.

IBM

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00000000		00	
00000001		01	
00000010		02	
00000011		03	
00000100		04	
00000101		05	
00000110		06	
00000111		07	
00001000		08	
00001001		09	
00001010		0A	
00001011		0B	
00001100		0C	
00001101		0D	
00001110		0E	
00001111		0F	
00010000		10	
00010001		11	
00010010		12	
00010011		13	
00010100		14	
00010101		15	
00010110		16	
00010111		17	
00011000		18	
00011001		19	
00011010		1A	
00011011		1B	
00011100		1C	
00011101		1D	
00011110		1E	
00011111		1F	
00100000		20	
00100001		21	
00100010		22	
00100011		23	
00100100		24	
00100101		25	
00100110		26	
00100111		27	
00101000		28	
00101001		29	
00101010		2A	
00101011		2B	
00101100		2C	
00101101		2D	
00101110		2E	
00101111		2F	
00110000		30	
00110001		31	
00110010		32	
00110011		33	
00110100		34	
00110101		35	
00110110		36	
00110111		37	
00111000		38	
00111001		39	
00111010		3A	
00111011		3B	
00111100		3C	
00111101		3D	
00111110		3E	
00111111		3F	
01000000	Blank	40	
01000001		41	
01000010		42	
01000011		43	
01000100		44	
01000101		45	
01000110		46	
01000111		47	
01001000		48	
01001001		49	
01001010	¢	4A	
01001011	.	4B	
01001100	<	4C	
01001101	(	4D	
01001110	+	4E	
01001111	|	4F	
01010000	&	50	
01010001		51	
01010010		52	
01010011		53	
01010100		54	
01010101		55	
01010110		56	
01010111		57	
01011000		58	
01011001		59	
01011010	!	5A	
01011011	$	5B	
01011100	*	5C	
01011101	)	5D	
01011110	;	5E	
01011111	¬	5F	
01100000	-	60	
01100001	/	61	
01100010		62	
01100011		63	
01100100		64	
01100101		65	
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011		6B	
01101100	%	6C	
01101101	–	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	
01111101	'	7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Figure 10-30. Forms Needed for Alternate Collating Sequence Specifications

Forms For Altering the Collating Sequence

Figure 10-30 illustrates two forms on which you must specify changes to the collating sequence. One form is the RPG II Control Card and File Description sheet; the other is the Translation Table and Alternate Collating Sequence Coding Sheet which is used for listing the actual changes in sequence. Both forms are used in conjunction with the RPG II Input, Output and Calculation sheets.

A letter *S* entered in column 26 of the RPG II control card notifies the program that additional information will be furnished to the program so that the collating sequence can be altered. All other columns contain the information that must normally be entered to process a job.

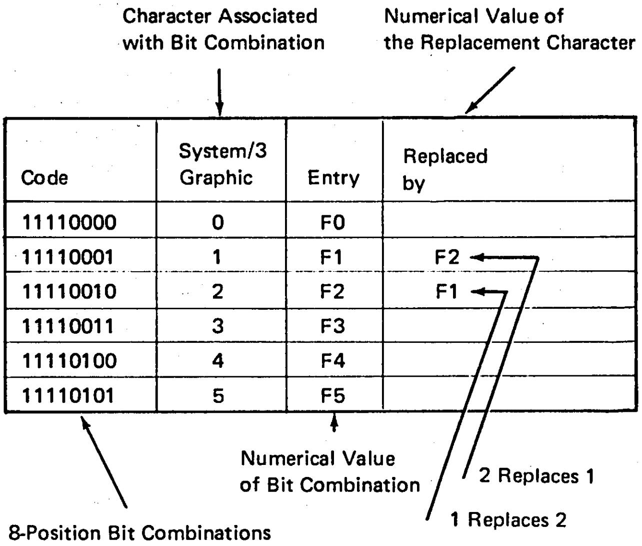
The Alternate Collating Sequence Coding Sheet lists 256 bit combinations along with their hexadecimal numerical values. As you learned from discussions of character structure, hexadecimal values are written in the form of two character values. One value represents the numerical value of the character's zone; the other represents the numerical value of the character's digit. The 64 printable graphics are listed beside the bit combinations and numerical values with which they are associated.

Coding a Change in Sequence

Each change in the collating sequence is specified in the *Replaced By* column on the coding sheet. In this column, place the hexadecimal value of the graphic whose position in the normal sequence is to be changed. The *character* corresponding to the hexadecimal value entered in the *Replaced By* column replaces the character which is presently associated with the bit combination shown on the same line.

Figure 10-31 illustrates entries made to change the normal collating sequence. Hexadecimal values entered on the second and third lines of the sample coding sheet reverse the order in which the numbers *1* and *2* are recognized by the computer.

Numerical values entered on the second line of the sample specify that the number 2 (hexadecimal value F2) replaces the number *1*. In other words, in the new sequence the number *2* is associated with the value F1 instead of the number *1*. Hexadecimal values on the third line specify that the number *1* (hexadecimal value F1) replaces the number *2*. These two specification lines cause *2* to come before *1* in the collating sequence (*0, 2, 1, 3*).



Character Associated with Bit Combination
Numerical Value of the Replacement Character
Numerical Value of Bit Combination
8-Position Bit Combinations
2 Replaces 1
1 Replaces 2

Code	System/3 Graphic	Entry	Replaced by
11110000	0	F0	
11110001	1	F1	F2
11110010	2	F2	F1
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	

Figure 10-31. Explanation of Alternate Collating Sequence Sheet

Effect of the Coded Change in Sequence

Any alternate collating sequence you specify is used temporarily. It is used only for the program which contains the alternate collating sequence specifications. Even more specifically, it is used in that program for operations which involve sequencing, such as checking sequence of records, comparing fields, or matching records.

You may think, according to specifications in Figure 10-31, that the character 2 read into the computer is *always* replaced by a 1. This is not true. The computer associates characters with the values you specify only before sequencing operations involving:

1. Compare operations on alphameric fields.

2. Matching or sequence checking match fields.

How does the computer keep track of the collating sequence to use? The computer keeps all your instructions for altering the sequence in storage. The area in storage which holds this information may be pictured as shown in Figure 10-32. These instructions combined with the pattern for normal sequence give the computer the correct collating sequence to use.

Numerical Value of Bit Combinations	Associated Characters	
	Normal Collating Sequence	Altered Collating Sequence
F0	0	0
F1	1	2
F2	2	1
F3	3	3
F4	4	4
F5	5	5
F6	6	6
F7	7	7

Figure 10-32. Storage Area Holding Alternate Collating Sequence Instruction

Consider the use of an altered sequence when determining which record to select for processing in a multifile program. The collating sequence has been changed so that *2* comes before *1.*

F1 and 1 ——→ F1 and 2

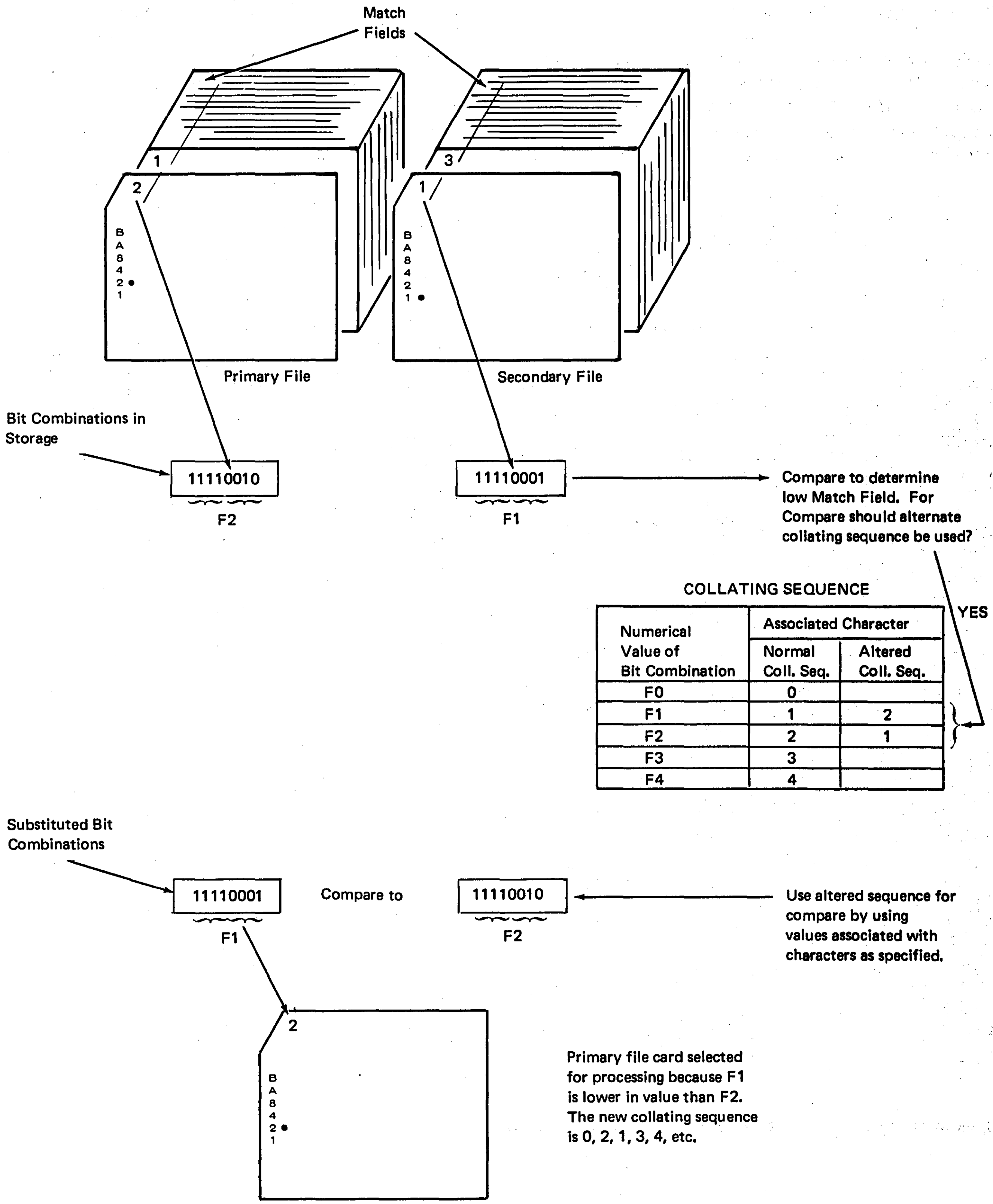
F2 and 2 ——→ F2 and 1

F3 and 3 ——→ F3 and 3

Figure 10-33 illustrates how the program uses the alternate sequence. Two cards are read into the read area. Just before the compare operation which is done to determine which match field has a lower value, the program checks to see if the characters used in the compare are affected by the alternate collating sequence instructions. They are. The character *1* normally associated with the value F1 is replaced by the character *2*; the character *2* normally associated with the value F2 is replaced by the character *1.*

When doing the compare, the program substitutes these values. For the match field having the character *2,* the program uses the bit combination whose value is F1 instead of the bit combination for F2. Similarly for the match field containing a *1,* the program uses the bit combination of F2 instead of the bit combination for F1. As a result of the compare, the primary card containing a 2 in the match field is chosen for processing. This card was chosen because F1 (now associated with character *2*) is lower in sequence than F2 (now associated with the character *1*).

After the compare, characters are again associated with values as assigned in the normal collating sequence.



Compare to determine low Match Field. For Compare should alternate collating sequence be used?

COLLATING SEQUENCE

Numerical Value of Bit Combination	Associated Character	
	Normal Coll. Seq.	Altered Coll. Seq.
F0	0	
F1	1	2
F2	2	1
F3	3	
F4	4	

Use altered sequence for compare by using values associated with characters as specified.

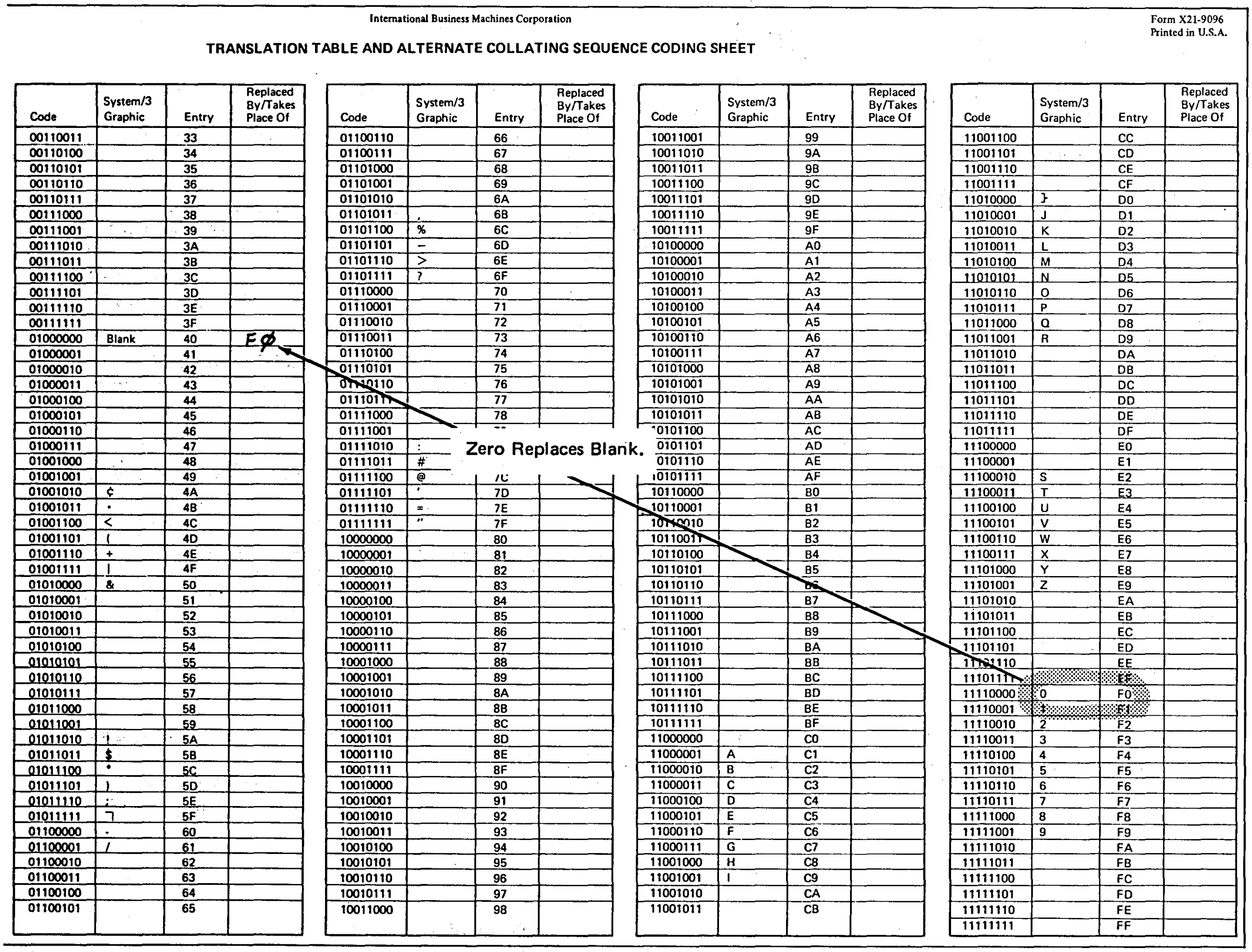
Primary file card selected for processing because F1 is lower in value than F2. The new collating sequence is 0, 2, 1, 3, 4, etc.

Figure 10-33. Using Alternate Collating Sequence (0, 1, 2, 3, 4, 9)

Coding Characters to be Equal

Entries can be made to allow two characters to occupy the same position in the collating sequence; that is, they are associated with the same numeric value. When two characters occupy the same position in the sequence, the computer recognizes one character as being the same as the other.

Figure 10-34 illustrates the specifications which allow a blank or zero to occupy the same position in an altered sequence (assume the field is alphameric). The hexadecimal value associated with the character blank is replaced by the hexadecimal value (F0) which is already associated with the zero. Because the zero and blank are associated with the same numerical value, they are recognized as the same character. Figure 10-35 shows why a field containing a blank is equal to a field containing a zero when the altered sequence is used.


Zero Replaces Blank.

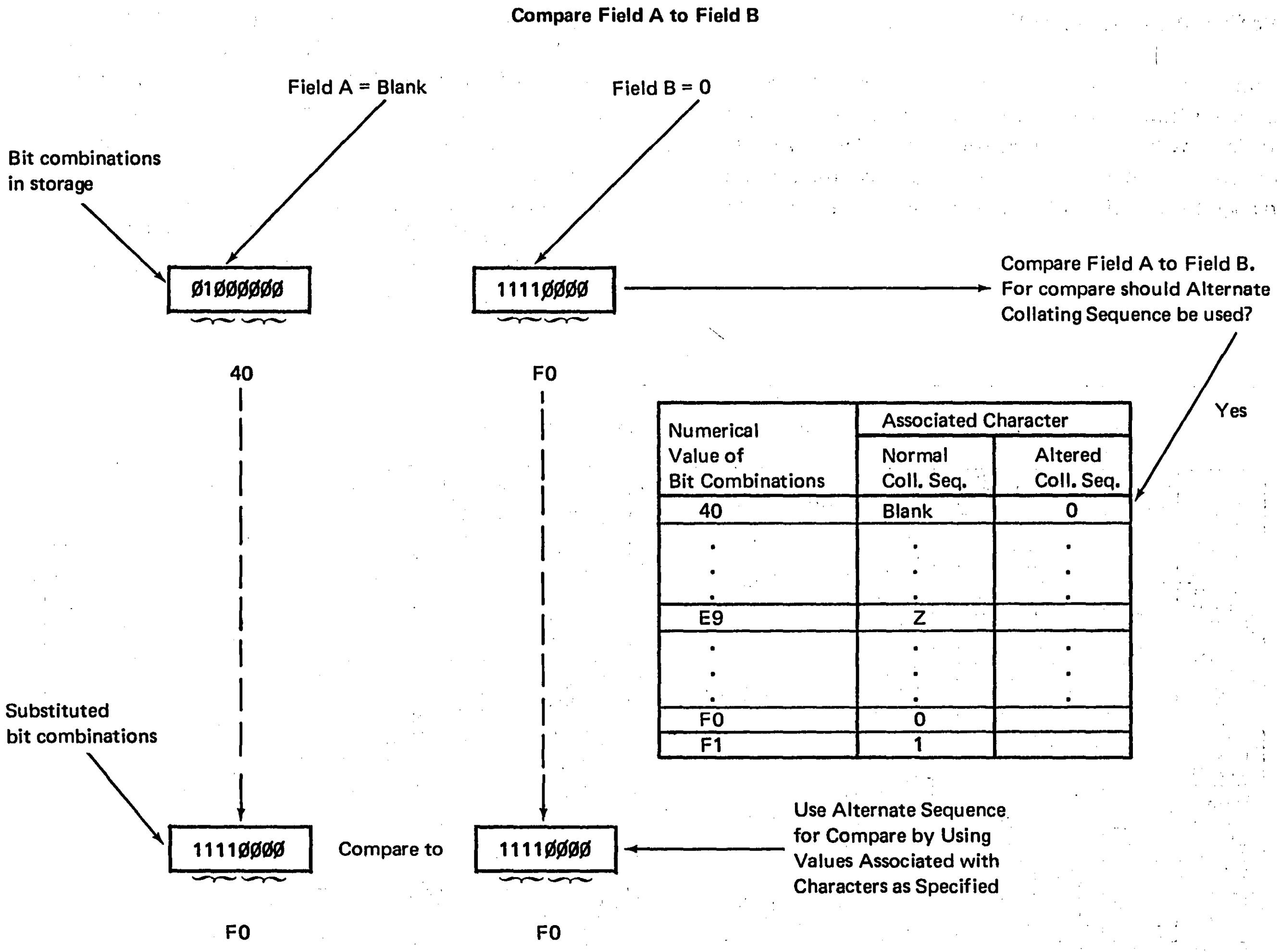
International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00110011		33		01100110		66		10011001		99		11001100		CC	
00110100		34		01100111		67		10011010		9A		11001101		CD	
00110101		35		01101000		68		10011011		9B		11001110		CE	
00110110		36		01101001		69		10011100		9C		11001111		CF	
00110111		37		01101010		6A		10011101		9D		11010000	}	D0	
00111000		38		01101011	,	6B		10011110		9E		11010001	J	D1	
00111001		39		01101100	%	6C		10011111		9F		11010010	K	D2	
00111010		3A		01101101	–	6D		10100000		A0		11010011	L	D3	
00111011		3B		01101110	>	6E		10100001		A1		11010100	M	D4	
00111100		3C		01101111	?	6F		10100010		A2		11010101	N	D5	
00111101		3D		01110000		70		10100011		A3		11010110	O	D6	
00111110		3E		01110001		71		10100100		A4		11010111	P	D7	
00111111		3F		01110010		72		10100101		A5		11011000	Q	D8	
01000000	Blank	40	FØ	01110011		73		10100110		A6		11011001	R	D9	
01000001		41		01110100		74		10100111		A7		11011010		DA	
01000010		42		01110101		75		10101000		A8		11011011		DB	
01000011		43		01110110		76		10101001		A9		11011100		DC	
01000100		44		01110111		77		10101010		AA		11011101		DD	
01000101		45		01111000		78		10101011		AB		11011110		DE	
01000110		46		01111001		[illegible]		10101100		AC		11011111		DF	
01000111		47		01111010	:	[illegible]		10101101		AD		11100000		E0	
01001000		48		01111011	#	[illegible]		10101110		AE		11100001		E1	
01001001		49		01111100	@	7C		10101111		AF		11100010	S	E2	
01001010	¢	4A		01111101	'	7D		10110000		B0		11100011	T	E3	
01001011	.	4B		01111110	=	7E		10110001		B1		11100100	U	E4	
01001100	<	4C		01111111	"	7F		10110010		B2		11100101	V	E5	
01001101	(	4D		10000000		80		10110011		B3		11100110	W	E6	
01001110	+	4E		10000001		81		10110100		B4		11100111	X	E7	
01001111	|	4F		10000010		82		10110101		B5		11101000	Y	E8	
01010000	&	50		10000011		83		10110110		B6		11101001	Z	E9	
01010001		51		10000100		84		10110111		B7		11101010		EA	
01010010		52		10000101		85		10111000		B8		11101011		EB	
01010011		53		10000110		86		10111001		B9		11101100		EC	
01010100		54		10000111		87		10111010		BA		11101101		ED	
01010101		55		10001000		88		10111011		BB		11101110		EE	
01010110		56		10001001		89		10111100		BC		11101111		EF	
01010111		57		10001010		8A		10111101		BD		11110000	0	F0	
01011000		58		10001011		8B		10111110		BE		11110001	1	F1	
01011001		59		10001100		8C		10111111		BF		11110010	2	F2	
01011010	!	5A		10001101		8D		11000000		C0		11110011	3	F3	
01011011	$	5B		10001110		8E		11000001	A	C1		11110100	4	F4	
01011100	*	5C		10001111		8F		11000010	B	C2		11110101	5	F5	
01011101	)	5D		10010000		90		11000011	C	C3		11110110	6	F6	
01011110	;	5E		10010001		91		11000100	D	C4		11110111	7	F7	
01011111	¬	5F		10010010		92		11000101	E	C5		11111000	8	F8	
01100000	-	60		10010011		93		11000110	F	C6		11111001	9	F9	
01100001	/	61		10010100		94		11000111	G	C7		11111010		FA	
01100010		62		10010101		95		11001000	H	C8		11111011		FB	
01100011		63		10010110		96		11001001	I	C9		11111100		FC	
01100100		64		10010111		97		11001010		CA		11111101		FD	
01100101		65		10011000		98		11001011		CB		11111110		FE	
												11111111		FF	

Figure 10-34. Specifying Blank Equal to Zero in New Collating Sequence


Compare Field A to Field B
Field A = Blank
Field B = 0
Bit combinations in storage
Ø1ØØØØØØ
1111ØØØØ
Compare Field A to Field B. For compare should Alternate Collating Sequence be used?
Yes
40
F0
Substituted bit combinations
1111ØØØØ
Compare to
1111ØØØØ
Use Alternate Sequence for Compare by Using Values Associated with Characters as Specified
F0
F0

Numerical Value of Bit Combinations	Associated Character	
	Normal Coll. Seq.	Altered Coll. Seq.
40	Blank	0
. . .	. . .	. . .
E9	Z	
. . .	. . .	. . .
F0	0	
F1	1	

Result: F0 is the same as F0
Fields are equal; Blank
is the same as zero.

Figure 10-35. Using Alternate Collating Sequence (Blank Equals Zero)

Example of the Coding of an Altered Sequence

Figure 10-36 shows a part of the normal collating sequence, and one of several ways in which the sequence can be changed. Arrows depict changes required in the positions of characters to alter the sequence as shown at the right side of the figure.

In like manner, arrows in Figure 10-37 show entries on the coding sheet which must be specified to alter the sequence. Note that letters *B* through *I* are to be repositioned to allow the at-symbol (@) to appear between letters *A* and *B*. Identical results could be achieved by repositioning the value for the letter *A* to the line above, making it correspond to bit combination 1100000.

To produce the sequence shown in Figure 10-36 and 10-37, the appropriate hexadecimal values must be specified in the *Replaced By* column beside each graphic involved in the change. Figure 10-38 shows the actual coding required to alter the sequence.

Notice that each number which is to be collated before the alphabetic character is assigned a hexadecimal value which has no graphic associated with it. These values have no associated graphics that could have been assigned to the values previously associated with numbers. This is not necessary, however, because these values have no associated graphics. When two graphics are involved in the change, then both must be assigned different values except when they are to be considered equal.

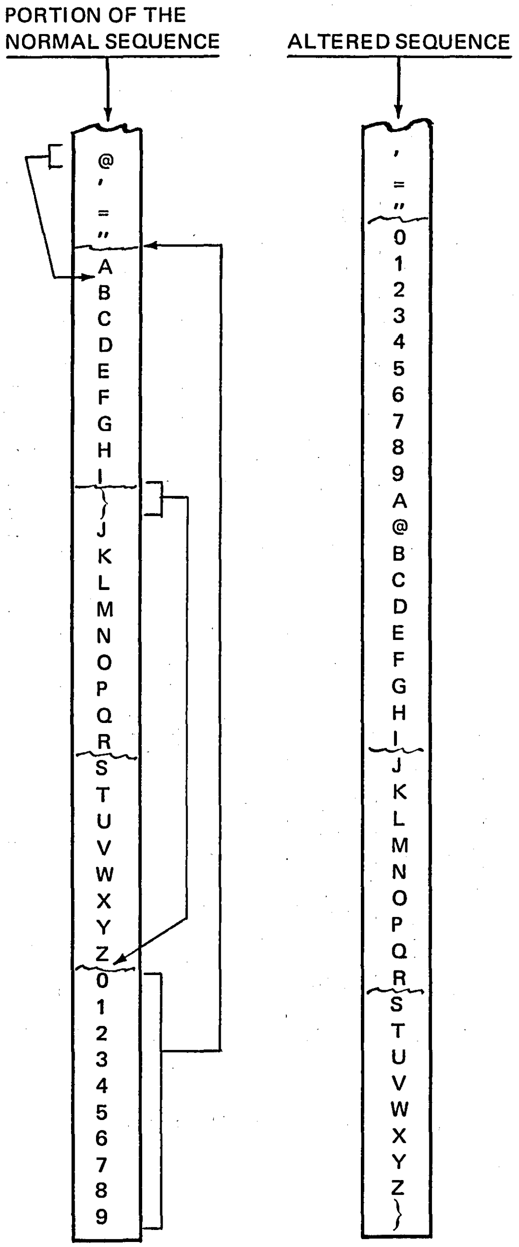
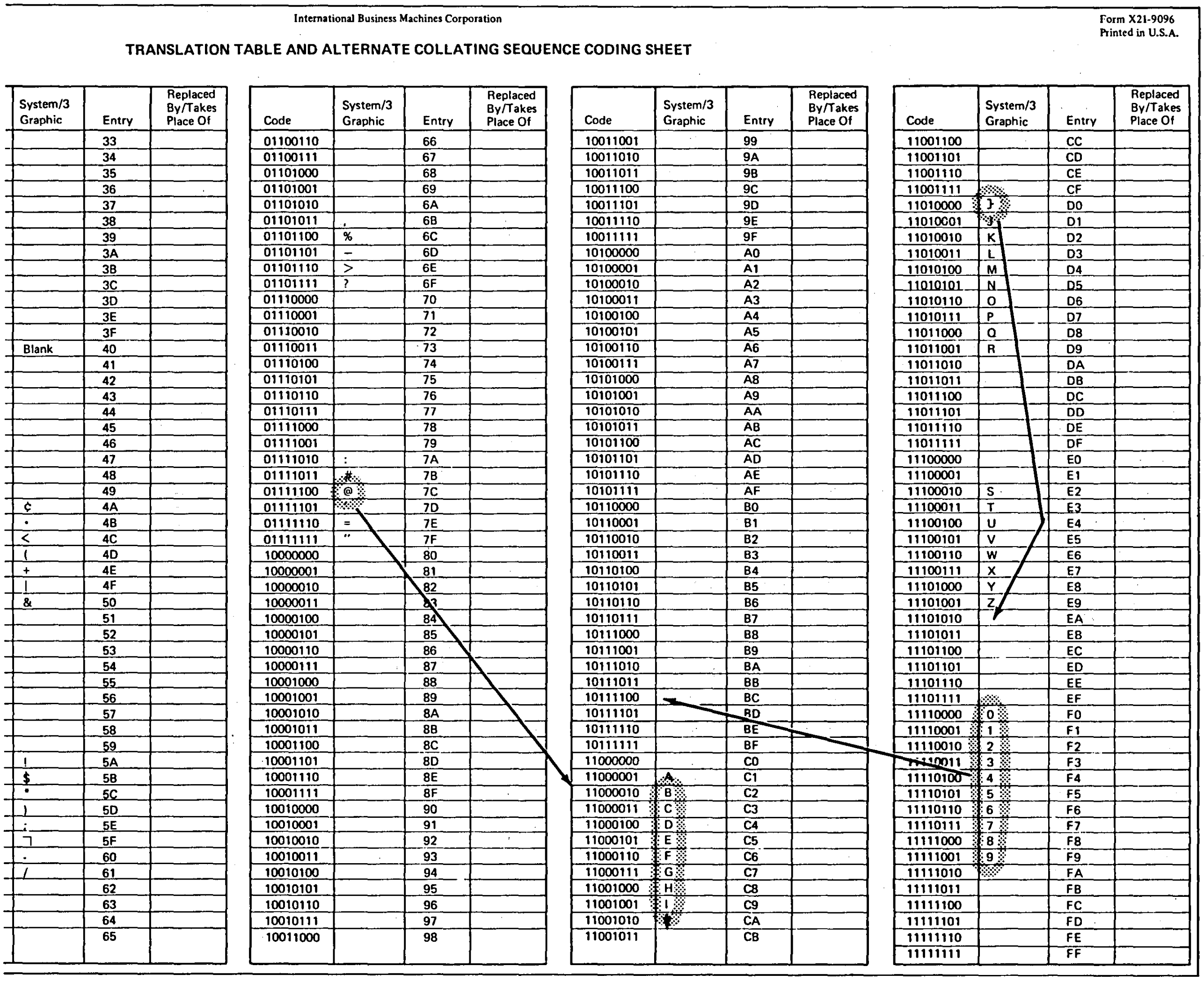

PORTION OF THE NORMAL SEQUENCE
@ ' = " A B C D E F G H I } J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9
ALTERED SEQUENCE
' = " 0 1 2 3 4 5 6 7 8 9 A @ B C D E F G H I J K L M N O P Q R S T U V W X Y Z }

Figure 10-36. Normal Sequence Versus Altered Sequence



International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

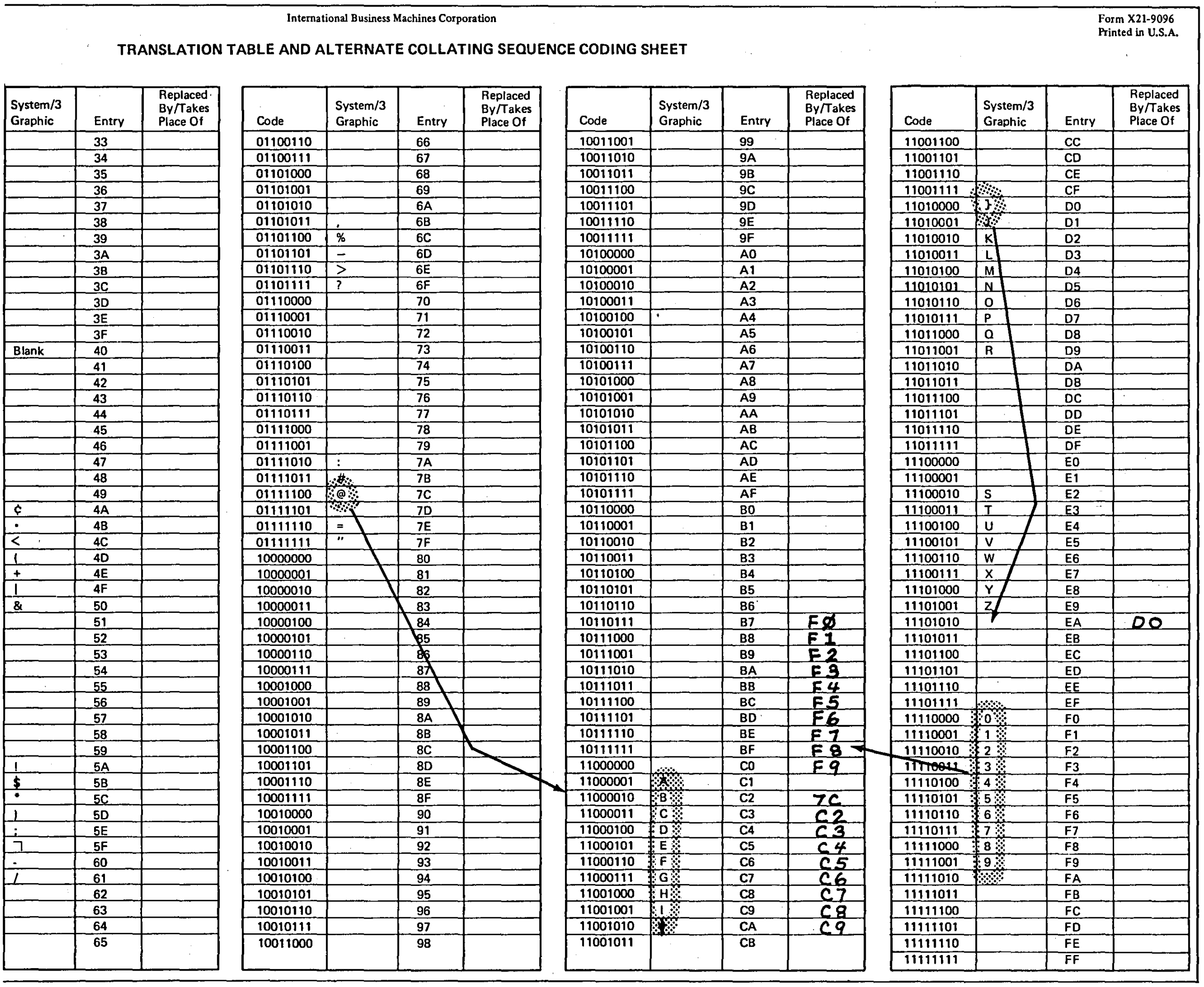
System/3 Graphic	Entry	Replaced By/Takes Place Of
	33	
	34	
	35	
	36	
	37	
	38	
	39	
	3A	
	3B	
	3C	
	3D	
	3E	
	3F	
Blank	40	
	41	
	42	
	43	
	44	
	45	
	46	
	47	
	48	
	49	
¢	4A	
.	4B	
<	4C	
(	4D	
+	4E	
|	4F	
&	50	
	51	
	52	
	53	
	54	
	55	
	56	
	57	
	58	
	59	
!	5A	
$	5B	
*	5C	
)	5D	
;	5E	
¬	5F	
-	60	
/	61	
	62	
	63	
	64	
	65	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011	,	6B	
01101100	%	6C	
01101101	–	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	
01111101	'	7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Figure 10-37. Changes Necessary for Altered Sequence



International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
		33	
		34	
		35	
		36	
		37	
		38	
		39	
		3A	
		3B	
		3C	
		3D	
		3E	
		3F	
	Blank	40	
		41	
		42	
		43	
		44	
		45	
		46	
		47	
		48	
		49	
	¢	4A	
	.	4B	
	<	4C	
	(	4D	
	+	4E	
	|	4F	
	&	50	
		51	
		52	
		53	
		54	
		55	
		56	
		57	
		58	
		59	
	!	5A	
	$	5B	
	*	5C	
	)	5D	
	;	5E	
	¬	5F	
	-	60	
	/	61	
		62	
		63	
		64	
		65	
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011	,	6B	
01101100	%	6C	
01101101	–	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	
01111101		7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	FØ
10111000		B8	F1
10111001		B9	F2
10111010		BA	F3
10111011		BB	F4
10111100		BC	F5
10111101		BD	F6
10111110		BE	F7
10111111		BF	F8
11000000		C0	F9
11000001	A	C1	
11000010	B	C2	7C
11000011	C	C3	C2
11000100	D	C4	C3
11000101	E	C5	C4
11000110	F	C6	C5
11000111	G	C7	C6
11001000	H	C8	C7
11001001	I	C9	C8
11001010		CA	C9
11001011		CB	
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	D0
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Figure 10-38. Coding for Altered Sequence

Recording Specifications for the Altered Sequence

After you have coded all specifications for the alternate collating sequence, you can record them so that they can be used by the computer. Records describing the alternate sequence are to be formatted as follows:

Positions	*Entries*
1-6	ALTSEQ (This entry allows the computer to recognize that this record is describing an alternate sequence.)
7-8	Blank
9-96	The hexadecimal values involved in changing the sequence.

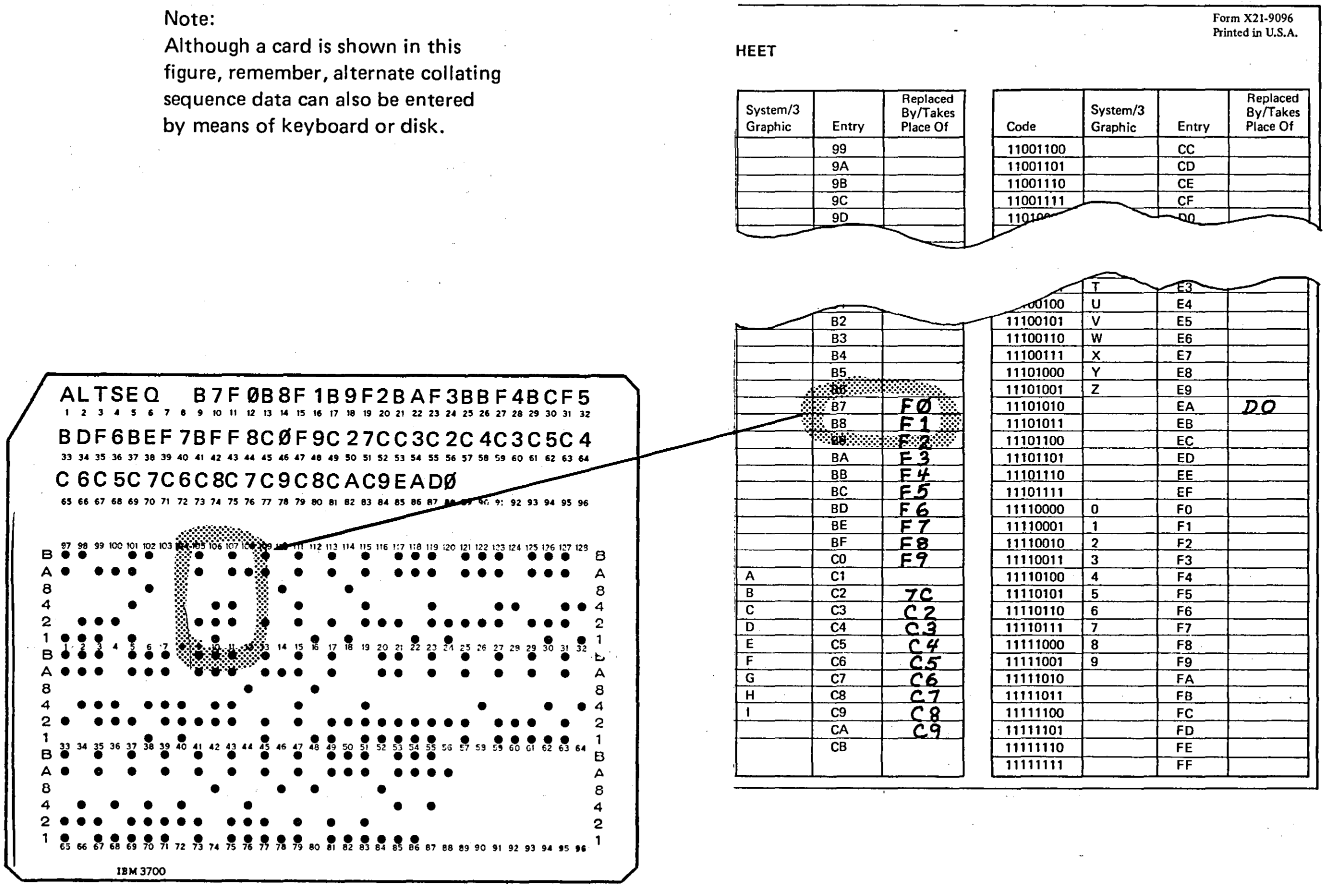
In positions 9-96, there are 22 groups of 4 positions. Each group (9-12, 13-16, etc.) must contain two hexadecimal values involved in changing the sequence. The first two positions of a group are for the hexadecimal value taken from the *Entry* column of the Alternate Collating Sequence Coding Sheet. The last two positions in a group are for the hexadecimal value taken from the *Replaced By* column of the coding sheet (Figure 10-39).

More than one record may be used to specify changes in collating sequence. However, each additional record must be formatted in the same way as the first.

The first blank appearing in positions 9-96 is recognized by the compiler as the end of the record. Consequently, blanks must not appear between pairs of hexadecimal values.

A record containing **ɓ (two asterisks and a blank) in positions 1 through 3 must precede the sequence records.

All records (except the RPG II control card) used for altering the collating sequence must follow RPG II specifications (or file translation specifications, when used) and must precede any tables being entered.



Note:
Although a card is shown in this figure, remember, alternate collating sequence data can also be entered by means of keyboard or disk.

```
ALTSEQ  B7FØB8F1B9F2BAF3BBF4BCF5
BDF6BEF7BFF8CØF9C27CC3C2C4C3C5C4
C6C5C7C6C8C7C9C8CAC9EADØ
```

HEET

Form X21-9096
Printed in U.S.A.

System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
	99		11001100		CC	
	9A		11001101		CD	
	9B		11001110		CE	
	9C		11001111		CF	
	9D		11010		D0	

System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
				T	E3	
			00100	U	E4	
	B2		11100101	V	E5	
	B3		11100110	W	E6	
	B4		11100111	X	E7	
	B5		11101000	Y	E8	
	B6		11101001	Z	E9	
	B7	FØ	11101010		EA	DØ
	B8	F1	11101011		EB	
	B9	F2	11101100		EC	
	BA	F3	11101101		ED	
	BB	F4	11101110		EE	
	BC	F5	11101111		EF	
	BD	F6	11110000	0	F0	
	BE	F7	11110001	1	F1	
	BF	F8	11110010	2	F2	
	C0	F9	11110011	3	F3	
A	C1		11110100	4	F4	
B	C2	7C	11110101	5	F5	
C	C3	C2	11110110	6	F6	
D	C4	C3	11110111	7	F7	
E	C5	C4	11111000	8	F8	
F	C6	C5	11111001	9	F9	
G	C7	C6	11111010		FA	
H	C8	C7	11111011		FB	
I	C9	C8	11111100		FC	
	CA	C9	11111101		FD	
	CB		11111110		FE	
			11111111		FF	

Figure 10-39. Punching Alternate Collating Sequence Cards

ALTERING THE STRUCTURE OF CHARACTERS

You learned in the discussion of character structure that each System/3 graphic is represented in the machine by a unique setting of eight bits; four zone bits and four digit bits. If any change is made to either the zone or digit bits, the entire character is changed. For example, if the *A* bit of the letter *M* is changed from on to off, the letter *M* becomes the letter *D* (Figure 10-40).

You can, of course, change a character before it is read into the computer by punching different zone punches on the card. But you can also change a character after it has been read. This is done by changing the zones of characters through the use of move zone operation codes.

Why would you ever want to change the zone of a character after it has been read? One common reason for changing zones is to deliberately change the sign of a field from positive to negative, or vice versa.

This is necessary when a numeric field read in from a special file has its sign in the high-order (leftmost) position of the field. Numeric fields are required to have the sign in the low-order (rightmost) position of the field. Thus, a numeric input field having its sign in the high-order position must have its sign moved to the low-order position. The move zone operations allow you to do this.

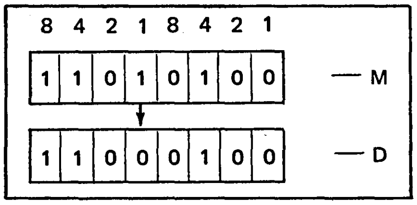


Figure 10-40. Changing Zones Changes Characters

How Move Zone Operations Work

Move zone operations involve only the zone portion of characters. The computer does not actually move the zone of one character to the zone portion of another. Rather, it changes a character by making its zone identical to the zone of the character which you indicate should serve as the model. The character serving as a model is not changed by the operation.

Thus, in order to use the move zone operations you must have:

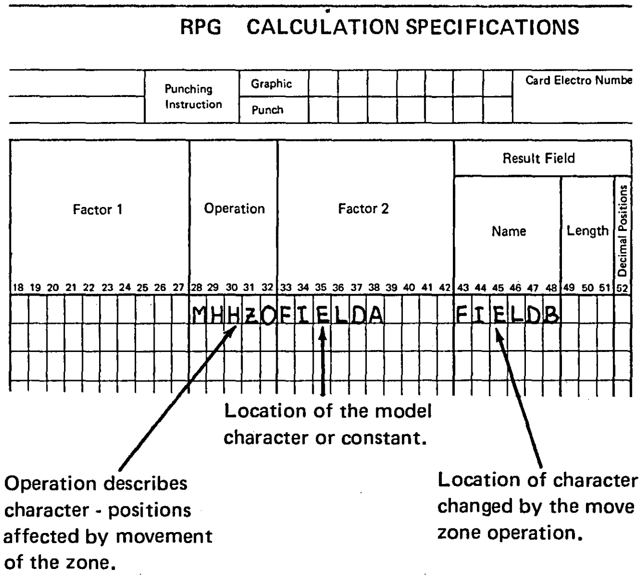
1. A character which needs to be changed.

2. A character that has the zone you want the changed character to have.

For example, if you want the low-order (rightmost) position of the field AMOUNT to be changed from a positive *5* to a negative *5* you must have a character to serve as a model whose zone portion is the same as the zone of a negative five.

Coding a Move Zone Operation

Figure 10-41 illustrates the way in which a move zone operation is coded. The name of the field containing the character to be changed must be entered in the Result Field. Either a constant of the name of the field which contains the model character must be entered in Factor 2. The move zone operation code is specified in the Operation columns (28-32). Any conditioning indicators you wish to use can be specified, but resulting indicators cannot be used.



Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions
	MHHZO	FIELDA	FIELDB		

Location of the model character or constant.

Operation describes character - positions affected by movement of the zone.

Location of character changed by the move zone operation.

Figure 10-41. Coding for a Move Zone Instruction

Differences in the Move Zone Operations

There are four different move zone operation codes available. Each code involves the zones of characters located in different positions; namely:

1. High-order positions in both Factor 2 and the Result Field.

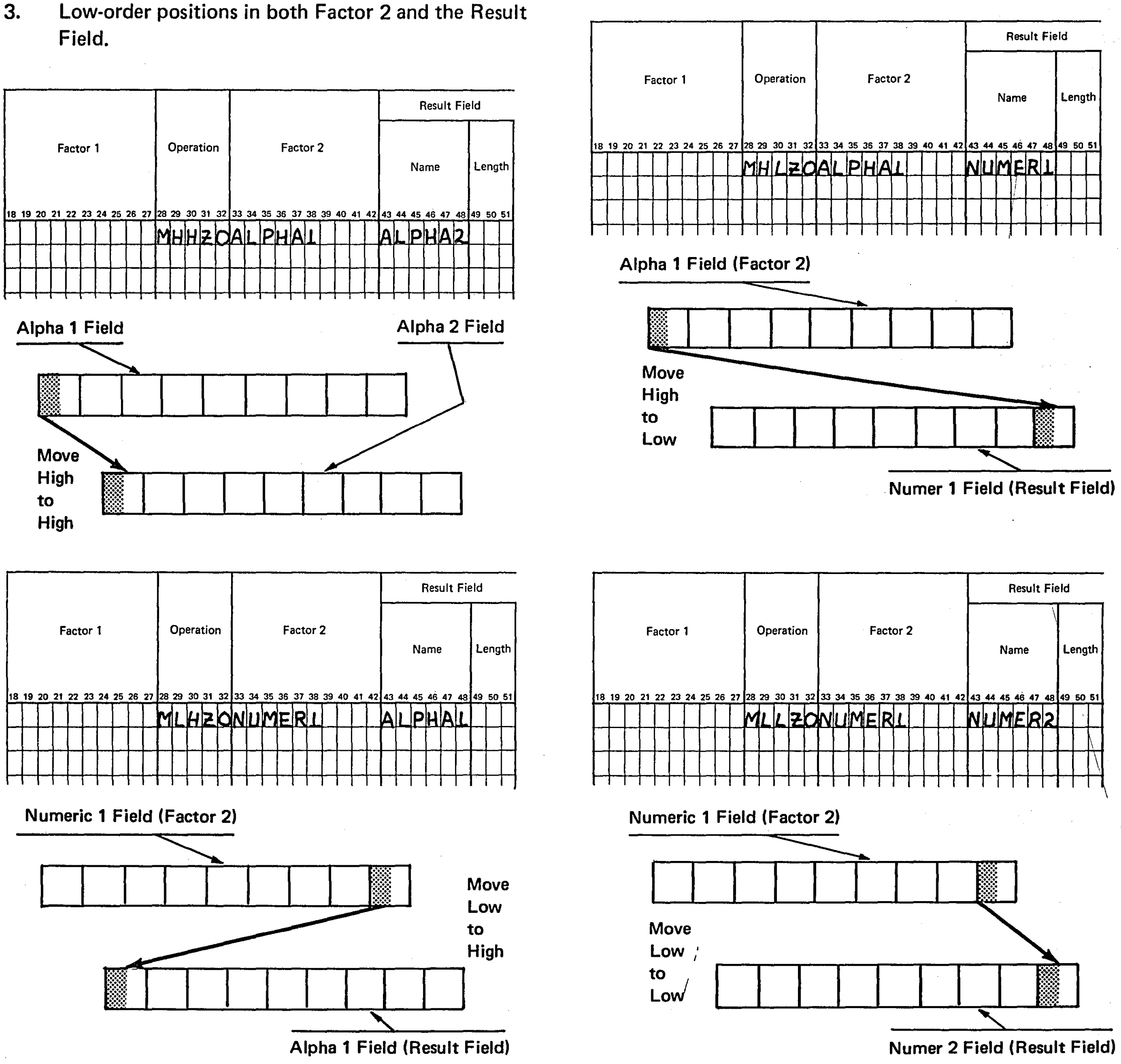
2. High-order position in Factor 2 and low-order in the Result Field.

3. Low-order positions in both Factor 2 and the Result Field.

4. Low-order position in Factor 2 and high-order in the Result Field.

Since only the zones of high and low-order characters in a field or constant are involved in the move zone operations, only the high or low-order positions of a field can be changed.

Figure 10-42 illustrates the ways in which the four operation codes affect the zone of a character in the Result Field.



Factor 1	Operation	Factor 2	Result Field Name	Result Field Length
	MHHZO	ALPHA1	ALPHA2	
	MHLZO	ALPHA1	NUMER1	
	MLHZO	NUMER1	ALPHA1	
	MLLZO	NUMER1	NUMER2	

Figure 10-42. Move Zone Operations

Move From High-Order Zone to High-Order Zone (MHHZO)

This operation code moves the zone of the high-order (leftmost) alphameric character in the constant or field entered in Factor 2 to the high-order alphameric character in the Result Field.

Move From Low-Order Zone to High-Order Zone (MLHZO)

This operation code moves the zone of the low-order (rightmost) character in the field or constant entered in Factor 2 to the high-order alphameric character in the Result Field. The Result Field must be alphameric; Factor 2 can be either numeric or alphameric.

Move From High-Order Zone to Low-Order Zone (MHLZO)

This operation code moves the zone of the high-order alphameric character in the constant or field entered in Factor 2 to the low-order rightmost character in the Result Field. Because of its high-order zone, Factor 2 must be an alphameric field. The Result Field can be either alphameric or numeric.

Move From Low-Order Zone to Low-Order Zone (MLLZO)

This operation code moves the zone of the low-order character in the field or constant entered in Factor 2 field to the low-order character in the Result Field. Both Factor 2 and the Result Field can be either numeric or alphameric.

Field Format and Move Zone Operations

As you read the description of each move zone operation, you probably noticed that special attention was given to the types of fields which can be used with each operation. Keep in mind that you cannot move from or to the high-order positions of a numeric field because the computer does not use the high-order zone of fields defined as numeric.

Which of the following move zone operations can be done if the two fields involved have formats as given below?

1. Alphameric to Alphameric: MHLZO

2. Alphameric to Numeric: MHHZO

3. Numeric to Alphameric: MLHZO

4. Numeric to Alphameric: MHHZO

5. Numeric to Numeric: MLHZO

6. Numeric to Numeric: MLLZO

Items *1, 3,* and *6* can be done. Items *2, 4,* and *5* cannot be done. Item *2* suggests that the zone of the high-order position in the numeric field be changed. The computer does not use high-order zone of numeric fields. Item *4* suggests that the zone of the high-order character is to serve as a model. It cannot because the computer does not work with the zones of high-order characters in a numeric field. Item *5* cannot be done because again it involves high-order positions of numeric fields.

Example of a Move Zone Operation

Now that you know how the various move zone operation codes work, let's see how they can be used to change the sign of the field, VALUE, from the high-order to the low-order position.

Naturally any field that has zones other than in the low-order position must be defined as alphameric if those zones are to be used by the computer. But if the field is to be involved in an arithmetic operation, it must be numeric.

To allow for both possibilities, you could define the field twice; once as alphameric and once as numeric. (Two unique field names are needed.) Another possibility is to define the field once as alphameric and then change it into a numeric field by moving it into a numeric field. This is what is done in the example (Figure 10-43).

Before doing any arithmetic operation, you must get the sign in the low-order position of a *numeric* field. You may want to first determine what the sign is by means of a TESTZ operation. Remember that TESTZ turns on the minus indicator when it finds the characters –, } , or *J* through *R* in the high-order position of the tested field. The specification in Figure 10-43, insert B, line 02, causes indicator 20 to turn on if the sign of the field is minus. If indicator 20 is on, the zone of VALUE, which is the minus sign to the computer, is moved to the low-order position of the AMOUNT field. If the field tested is plus, no zone is moved because a numeric field having no minus sign is automatically assumed to be positive.

Notice that the MHLZO (Move High to Low Zone) operation code was used to change the zone of the low-order position of the AMOUNT field by giving it the same zone as the high-order position of VALUE.

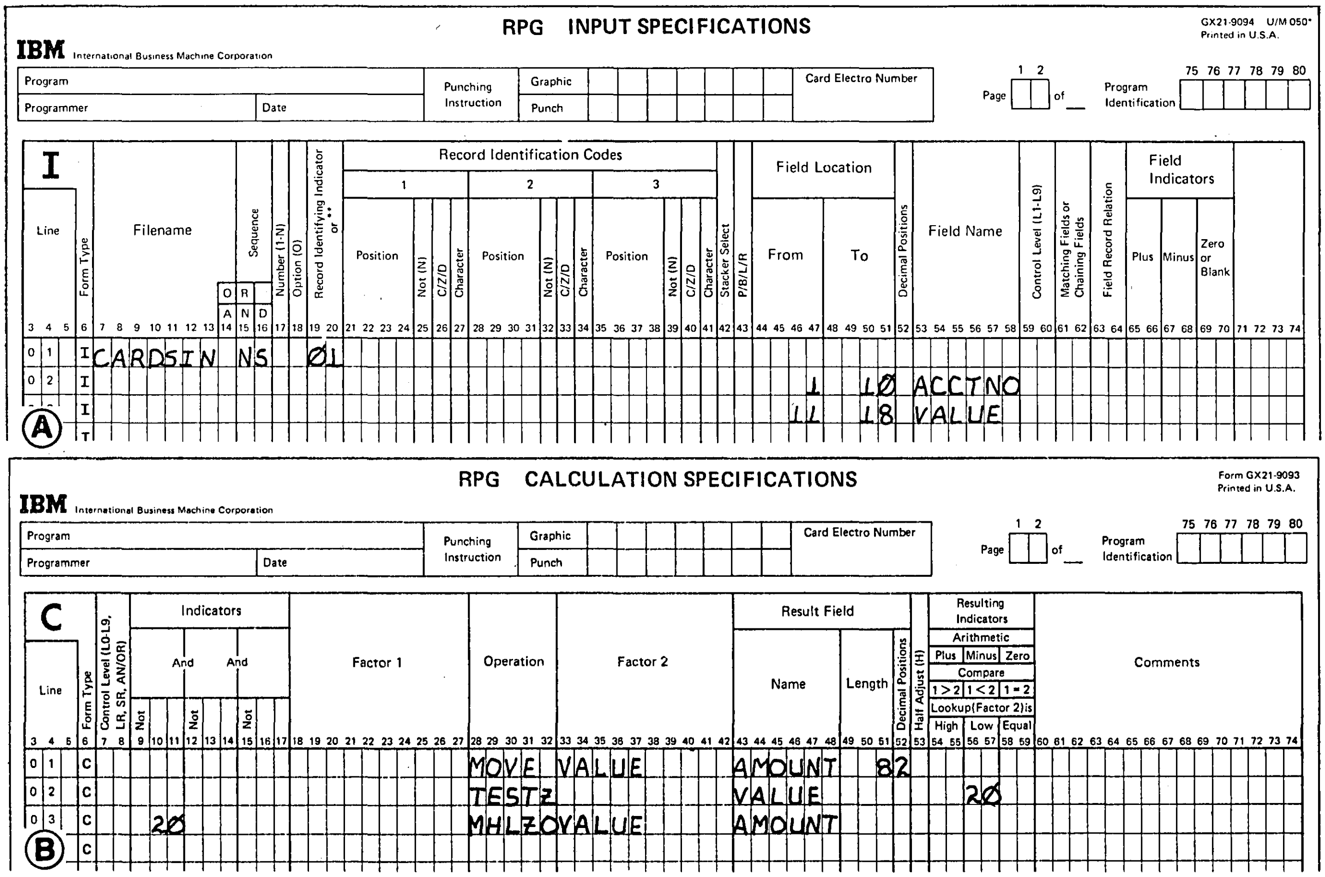
Note: This example can also be accomplished without using a TESTZ operation. First, move the zone of the high-order position of the field VALUE to the low-order position. This puts the sign in the low-order position. The alphameric VALUE field can then be moved to the numeric AMOUNT field and the arithmetic operation can be performed.

Choosing the Model Character for Factor 2

Before specifying a move zone operation, you must have a character designated in Factor 2 whose zone will give the desired zone in the Result Field.

Usually you will use move zone operations to change the signs of fields. Using any numbers in Factor 2 will produce a positive character in the Result Field. Using any one of the character } or *J-R* in Factor 2 will give you a negative character. The - (minus, X'60') character does not work to make a character negative using the move zone operation. Remember that negative numbers are punched with a *B* punch (minus sign) over the number. The punch combinations of negative numbers have the same numeric value in the computer as *J-R*. Thus, when you specify that the zone of a character should be made like the zone of *J-R*, you will get a minus character. See *Character Structure* for more information.

Use Figure 10-28 as a guide for selecting the zone which will produce the desired change.



IBM International Business Machines Corporation

RPG INPUT SPECIFICATIONS

GX21-9094 U/M 050* Printed in U.S.A.

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Field Location From	Field Location To	Field Name
01	I	CARDSIN	NS	Ø1			
02	I				1	1Ø	ACCTNO
	I				11	18	VALUE
	I						

Ⓐ

IBM International Business Machines Corporation

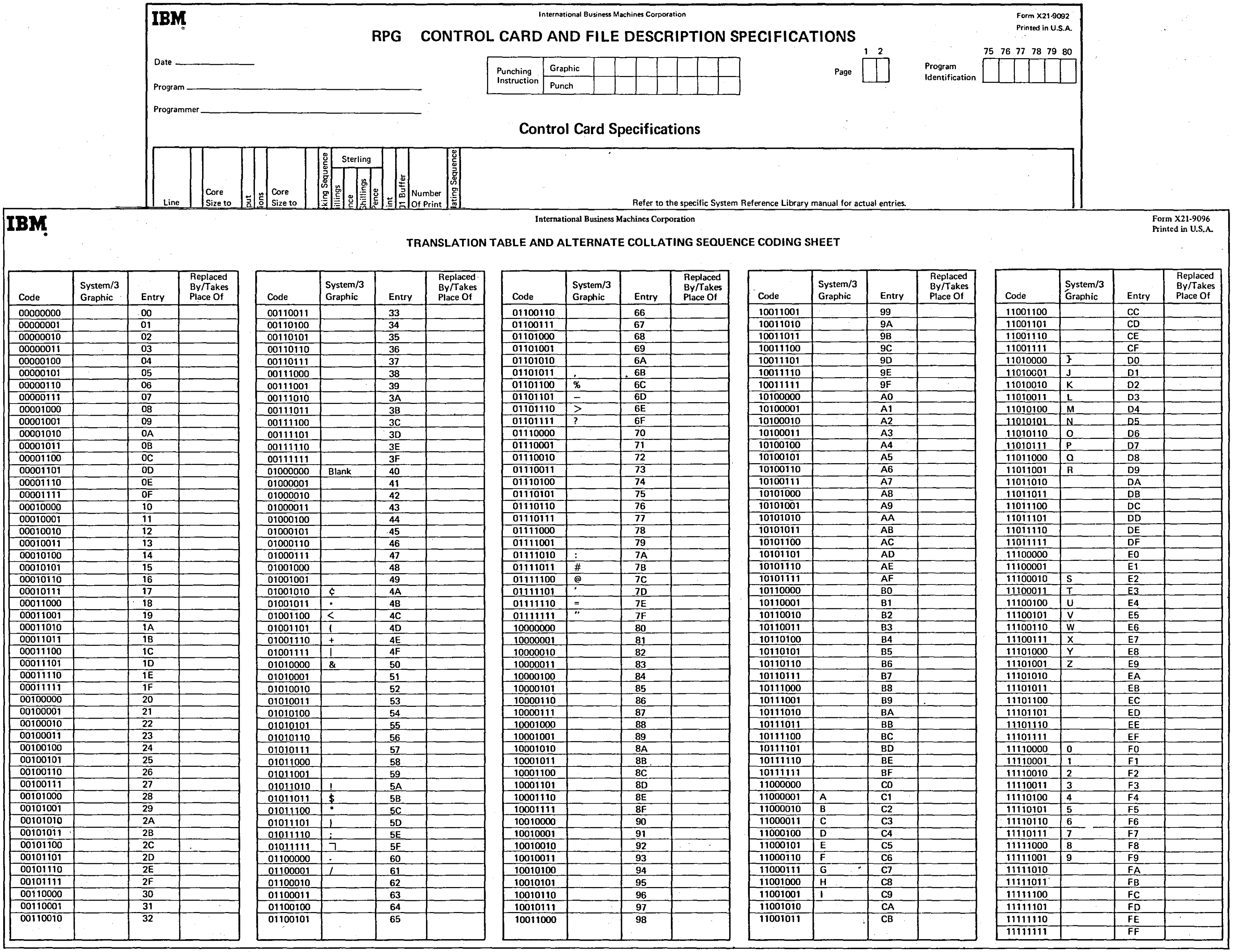
RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Resulting Indicators Minus
01	C			MOVE	VALUE	AMOUNT	8	2	
02	C			TESTZ		VALUE			2Ø
03	C	2Ø		MHLZO	VALUE	AMOUNT			
	C								

Ⓑ

Figure 10-43. Using Move Zone Operations to Change the Sign of a Field



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ______________

Program ______________

Programmer ______________

Punching Instruction	Graphic							
	Punch							

Page 1 2 □□

Program Identification 75 76 77 78 79 80 □□□□□□

Control Card Specifications

Line | Core Size to | put | ons | Core Size to | king Sequence | Sterling | illings | nce | Shillings | Pence | int | 01 Buffer | Number Of Print | lating Sequence

Refer to the specific System Reference Library manual for actual entries.

IBM

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00000000		00		00110011		33		01100110		66		10011001		99		11001100		CC	
00000001		01		00110100		34		01100111		67		10011010		9A		11001101		CD	
00000010		02		00110101		35		01101000		68		10011011		9B		11001110		CE	
00000011		03		00110110		36		01101001		69		10011100		9C		11001111		CF	
00000100		04		00110111		37		01101010		6A		10011101		9D		11010000	}	D0	
00000101		05		00111000		38		01101011	,	6B		10011110		9E		11010001	J	D1	
00000110		06		00111001		39		01101100	%	6C		10011111		9F		11010010	K	D2	
00000111		07		00111010		3A		01101101	–	6D		10100000		A0		11010011	L	D3	
00001000		08		00111011		3B		01101110	>	6E		10100001		A1		11010100	M	D4	
00001001		09		00111100		3C		01101111	?	6F		10100010		A2		11010101	N	D5	
00001010		0A		00111101		3D		01110000		70		10100011		A3		11010110	O	D6	
00001011		0B		00111110		3E		01110001		71		10100100		A4		11010111	P	D7	
00001100		0C		00111111		3F		01110010		72		10100101		A5		11011000	Q	D8	
00001101		0D		01000000	Blank	40		01110011		73		10100110		A6		11011001	R	D9	
00001110		0E		01000001		41		01110100		74		10100111		A7		11011010		DA	
00001111		0F		01000010		42		01110101		75		10101000		A8		11011011		DB	
00010000		10		01000011		43		01110110		76		10101001		A9		11011100		DC	
00010001		11		01000100		44		01110111		77		10101010		AA		11011101		DD	
00010010		12		01000101		45		01111000		78		10101011		AB		11011110		DE	
00010011		13		01000110		46		01111001		79		10101100		AC		11011111		DF	
00010100		14		01000111		47		01111010	:	7A		10101101		AD		11100000		E0	
00010101		15		01001000		48		01111011	#	7B		10101110		AE		11100001		E1	
00010110		16		01001001		49		01111100	@	7C		10101111		AF		11100010	S	E2	
00010111		17		01001010	¢	4A		01111101	'	7D		10110000		B0		11100011	T	E3	
00011000		18		01001011	.	4B		01111110	=	7E		10110001		B1		11100100	U	E4	
00011001		19		01001100	<	4C		01111111	"	7F		10110010		B2		11100101	V	E5	
00011010		1A		01001101	(	4D		10000000		80		10110011		B3		11100110	W	E6	
00011011		1B		01001110	+	4E		10000001		81		10110100		B4		11100111	X	E7	
00011100		1C		01001111	|	4F		10000010		82		10110101		B5		11101000	Y	E8	
00011101		1D		01010000	&	50		10000011		83		10110110		B6		11101001	Z	E9	
00011110		1E		01010001		51		10000100		84		10110111		B7		11101010		EA	
00011111		1F		01010010		52		10000101		85		10111000		B8		11101011		EB	
00100000		20		01010011		53		10000110		86		10111001		B9		11101100		EC	
00100001		21		01010100		54		10000111		87		10111010		BA		11101101		ED	
00100010		22		01010101		55		10001000		88		10111011		BB		11101110		EE	
00100011		23		01010110		56		10001001		89		10111100		BC		11101111		EF	
00100100		24		01010111		57		10001010		8A		10111101		BD		11110000	0	F0	
00100101		25		01011000		58		10001011		8B		10111110		BE		11110001	1	F1	
00100110		26		01011001		59		10001100		8C		10111111		BF		11110010	2	F2	
00100111		27		01011010	!	5A		10001101		8D		11000000		C0		11110011	3	F3	
00101000		28		01011011	$	5B		10001110		8E		11000001	A	C1		11110100	4	F4	
00101001		29		01011100	*	5C		10001111		8F		11000010	B	C2		11110101	5	F5	
00101010		2A		01011101	)	5D		10010000		90		11000011	C	C3		11110110	6	F6	
00101011		2B		01011110	;	5E		10010001		91		11000100	D	C4		11110111	7	F7	
00101100		2C		01011111	¬	5F		10010010		92		11000101	E	C5		11111000	8	F8	
00101101		2D		01100000	-	60		10010011		93		11000110	F	C6		11111001	9	F9	
00101110		2E		01100001	/	61		10010100		94		11000111	G	C7		11111010		FA	
00101111		2F		01100010		62		10010101		95		11001000	H	C8		11111011		FB	
00110000		30		01100011		63		10010110		96		11001001	I	C9		11111100		FC	
00110001		31		01100100		64		10010111		97		11001010		CA		11111101		FD	
00110010		32		01100101		65		10011000		98		11001011		CB		11111110		FE	
																11111111		FF	

Figure 10-44. Forms Needed for File Translation Specifications

TRANSLATING CHARACTERS

In the previous discussion, you learned that the program can alter the structure of characters by moving zones. But, through the file translation function of the RPG II language, it can do even more. It can translate one character into another.

The translating function is known as file translation because characters can be translated either when they are read in or before they are recorded in the output file. The program acts like an interpreter. Just as a human interpreter translates languages (a word in German for a word in English), the computer translates characters by replacing one character with another.

Need for File Translation

Think of the use for file translation when translating codes. Codes are often used as a security measure to prevent access to classified information. Information is recorded on cards in coded form. In order to process the information, it must be decoded. A coded character must be replaced by the corresponding decoded character.

For example, a firm which keeps all information classified uses the characters in the word FITZGERALD as a code for the numbers 0 through 9. *F* is the code for zero, *I* for one, etc. When recorded on a card, the number 1432 appears as IGZT. If a field containing IGZT is read into the computer and used in arithmetic operations, results received are wrong. IGZT must first be decoded, or translated into 1432.

Specifying File Translation

Specifications for file translation are identical to those used to alter the collating sequence.

Forms Used for a File Translation

Figure 10-44 shows the forms on which you must specify the way in which files are to be translated. One form consists of the RPG II Control Card and File Description sheet; the other consists of the Translation Table and Alternate Collating Sequence Coding Sheet for listing the characters to be translated. Both forms are used in conjunction with the RPG II Input, Output, and Calculation sheets.

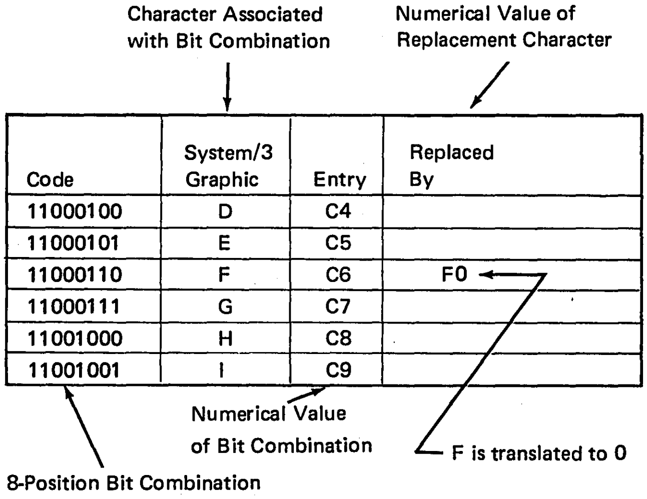
Only column 43 in the RPG II control card relates to the change in sequence. A letter *F* entered in column 43 notifies the computer that additional information furnished for translating files. All other columns contain the information that must normally be entered for a program.

The Translation Table and Alternate Collating Sequence Coding Sheet lists 256 bit combinations along with their hexadecimal numerical values. You learned from discussions of character structure that the left-hand number in the hexadecimal value represents the numerical value of the character's zone and the right-hand number represents the numerical value of the character's digit. The 64 printable characters are listed beside the bit combination and hexadecimal values with which they are associated.

Coding the Translation

Each character that will be affected during the translation of a specified file must be identified on the coding sheet. In the column entitled *Replaced By,* enter the hexadecimal value of the character which is to replace the character presently associated with the bit combination shown. This means that the character associated with the value found in the *Entry* column will be translated into the character associated with the value entered in the *Replaced By* column.

Figure 10-45 illustrates the entry made on the coding sheet to translate a character. If an input file is to be translated, this entry means that the letter *F* will be translated as the number *0* (F0 is the hexadecimal value associated with 0).


Character Associated with Bit Combination
Numerical Value of Replacement Character
Numerical Value of Bit Combination
F is translated to 0
8-Position Bit Combination

Code	System/3 Graphic	Entry	Replaced By
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	F0
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	

Figure 10-45. Explanation of File Translation Coding Sheet

If the output file is to be translated, this entry means that the number *0* will be translated back into an *F* before being written out. You can think of the character associated with the value in the *Entry* column as being the character read in or printed out. On the other hand, the character associated with the values in the *Replaced By* column is the character represented in the machine (Figure 10-46).

Differences Between File Translation and Alternate Collating Sequence

Because of the similarity of entries used in coding an alternate collating sequence and a file translation, these functions may seem identical. They are not, however. The difference occurs in the way the program works with the characters involved.

When alternate collating sequence is used, the characters are altered only temporarily for sequencing operations. The original bit combination of the character, obtained from the punch combination for that character, is not changed. Temporary substitution of another bit combination is done instead.

For file translation, bit combinations are actually changed. As a result, one character is changed (translated) into another. This translation occurs before your program instructions are executed.

What Files Should Be Translated?

Any input files which contain information recorded in coded form should be translated if correct results are to be obtained. All characters which you specify to be translated are translated whenever they are encountered. This means if you specify an *F* to be translated to *0*, all *F*'s read in will be translated. When there are several other fields on the cards in addition to the one containing coded information, remember all characters specified to be translated are translated regardless of fields.

When printing or punching information out, you may or may not find it necessary to specify file translation for the output files. If you have translated (decoded) your input file, you should translate information back into coded form before it is written or punched out. If all *F*'s are translated as *0*'s when read in, then all *0*'s should be translated to *F*'s before they are put out. Keep in mind that only characters which you specify are involved in the retranslation.

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

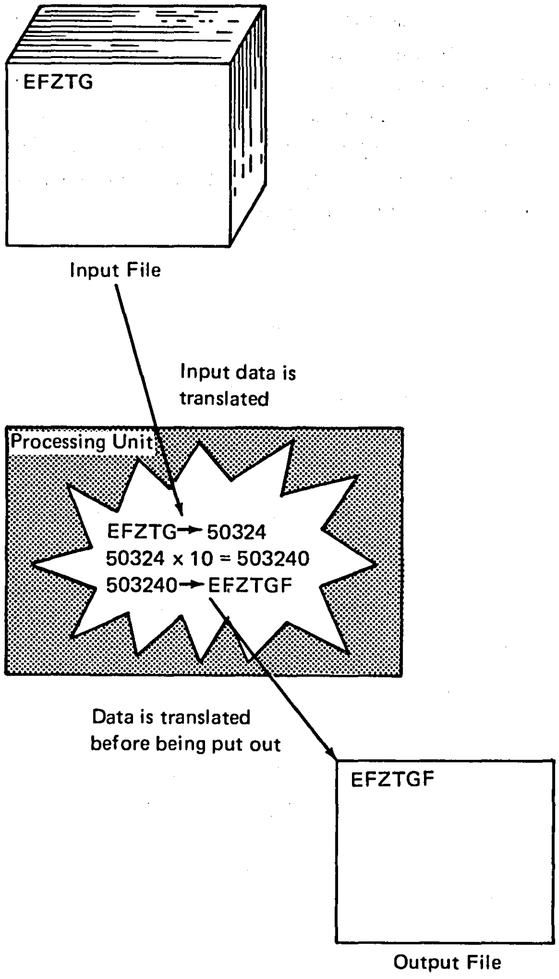
TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

	System/3 Graphic	Entry	Replaced By/Takes Place Of
1		33	
		34	
		35	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
		6B	
		6C	
		6D	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	F7
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	F9
11000101	E	C5	F5
11000110	F	C6	FØ
11000111	G	C7	F4
11001000	H	C8	
11001001	I	C9	F1
11001010		CA	
11001011		CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	F8
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	F6
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	F2
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	F3
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

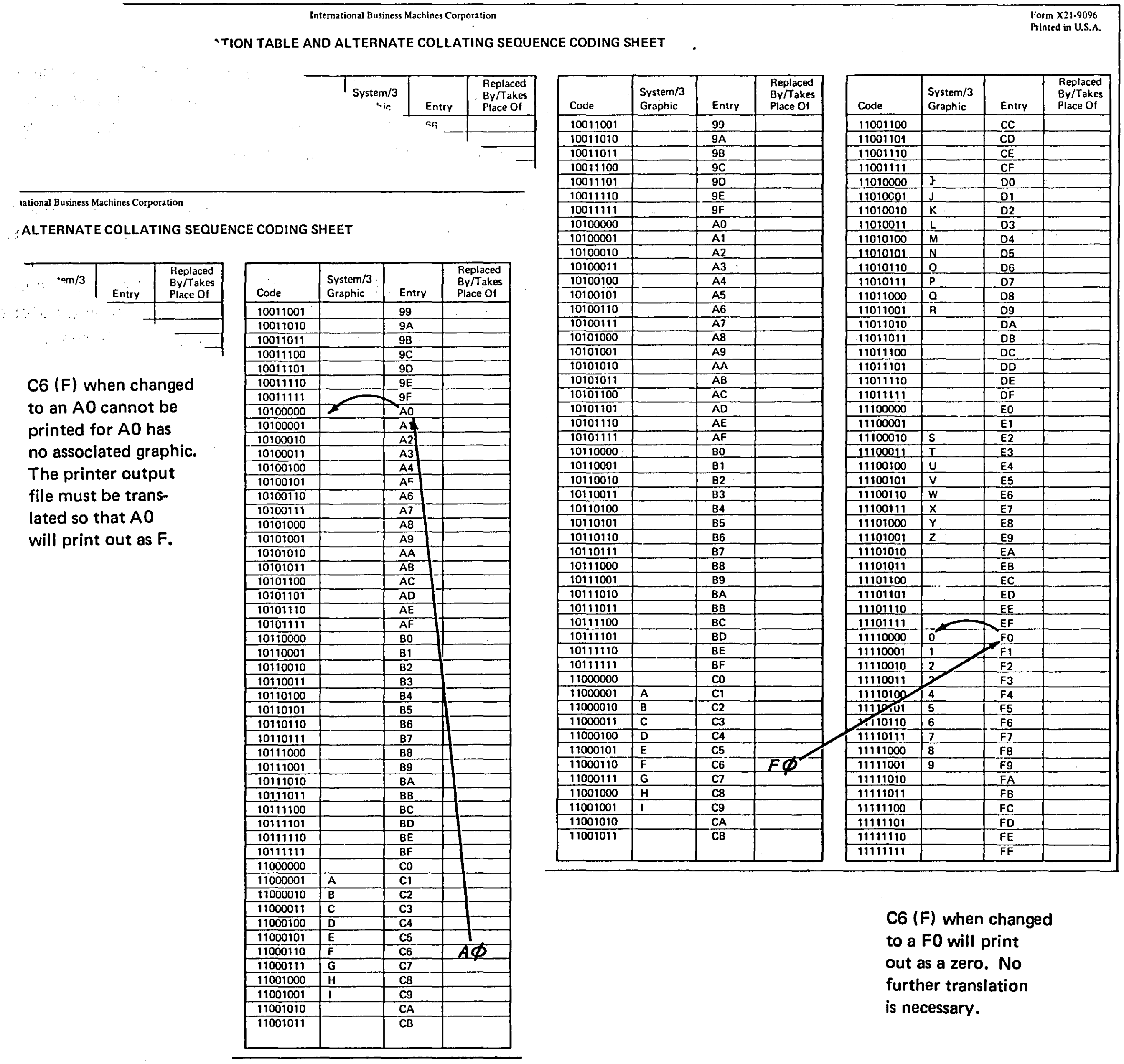


File Translation specifications used for translating both input and output files as follows:

F	I	T	Z	G	E	R	A	L	D
0	1	2	3	4	5	6	7	8	9

Figure 10-46. File Translation

If you do not specify file translation for output files, information is put out exactly as it is in the machine. If you do not intend to translate card or printer output files, be certain that all characters from the input file are translated into a value associated with a printable graphic. Any hexadecimal value which does not have an associated graphic cannot be written or punched out (Figure 10-47). If an unprintable graphic is specified to be put out, a blank appears in its place.



International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

ᐧTION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

System/3 ᐧic	Entry	Replaced By/Takes Place Of
	C6	

ıational Business Machines Corporation

ALTERNATE COLLATING SEQUENCE CODING SHEET

ᐧem/3	Entry	Replaced By/Takes Place Of

C6 (F) when changed to an A0 cannot be printed for A0 has no associated graphic. The printer output file must be translated so that A0 will print out as F.

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	AØ
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	FØ
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

C6 (F) when changed to a F0 will print out as a zero. No further translation is necessary.

Figure 10-47. Printable Graphics

Recording Specifications for the Translation Table

After you have written all specifications for file translation, you can record them so that they can be entered into the system. Records containing these specifications must be formatted as follows:

Positions	*Entry*
1-6 or	*FILES
1-8	A filename
7-8	Blank if not required
9-96	Numerical values involved in translating characters

If all files (both input and output) are to be translated, use the entry *FILES in positions 1 through 6. If only one file is translated, use that filename in positions 1 through 8. If several, but not all files, are to be translated, you must format separate records for each file.

In positions 9 through 96, there are 22 groups of four positions. Each group (9-12, 13-16, etc.) must contain two hexadecimal values involved in the translating of *one* character. The first two positions of the group are for the hexadecimal value taken from the *Entry* column of the Translation Table and Alternate Collating Sequence Coding Sheet. The last two positions are for the hexadecimal value taken from the *Replaced By* column of the coding sheet.

More than one record can be used to specify the characters which must be translated. However, each additional record must be formatted in the same way as the first. All records for one file must be grouped together. An error will occur if four records are entered in the following order:

1. FILEA

2. FILEA

3. FILEB

4. FILEA

Also, the first blank appearing in positions 9 through 96 is recognized by the computer as the end of the translation specifications. Consequently blanks should not appear between pairs of hexadecimal values.

A record containing **ƀ (two asterisks and a blank) in positions 1 through 3 must precede the file translation records.

All records used for file translation except the RPG II control card must follow RPG II input, calculation, and output specifications and must precede any tables or alternate collating sequence records used.

Character Structure

1. Into what two portions may every column of a 96-column card and every byte in storage and on disk be divided?

2. Do all characters that have an *A* zone punched in the zone portion of a 96-column card have the same zone representation in storage? Why or why not?

3. Calculate the numerical value of each of the following binary numbers as recorded in one byte of storage:
   a. 11000100.
   b. 11010101.
   c. 11101000.
   d. 11110011.

4. Express the numerical value of the bytes shown in Question 3 as a pair of numbers (hexadecimal value), rather than as a single value.

Collating Sequence of Characters

5. What does the computer use to determine the collating sequence of characters?

6. Arrange the following characters in ascending collating sequence. Arrange the same characters in ascending collating sequence by zone and digit.

Character	Hexadecimal Value	Numerical Value
C	C3	195
/	61	97
P	D7	215
J	D1	209
*	5C	92
T	E3	227
R	D9	217
4	F4	245
&	50	80
9	F9	249
0	F0	240

Altering the Collating Sequence

7. Fill in the Alternate Collating Sequence Coding Sheet to:
   a. Insert a Ü between U and V (use the # sign to represent Ü).
   b. Make a blank fall in the same sequence as zero.

   Show how this information would be recorded in an alternate collating sequence record.

8. In what RPG II operations is the alternate collating sequence used?

9. Where is the sign located in a numeric field?

Altering the Structure of Characters

10. The TESTZ operation checks the zones of:
    a. any position in a field.
    b. only the low-order position in the field.
    c. only the high-order position in a field.

11. A field may be alphameric for *any* move zone operations. Check those fields (Factor 2, Result) which *can be* numeric for the following move zone operations:

	Operation	Factor 2	Result
a.	MHLZO		
b.	MLHZO		
c.	MLLZO		
d.	MHHZO		

12. Code the calculation specifications to make the contents of a positive numeric AMTDUE field negative.

Translating Characters

13. What is the difference between the way the computer works with characters involved in an alternate collating sequence and the way it works with characters involved in file translation?

14. Fill in the coding forms to translate *A*'s to *1*'s and *B*'s to *3*'s. Show how these specifications would be recorded in a translation table record when all files are to be translated.

1. Zone and digit.

2. All characters which have the same zone punch in a 96-column card do not necessarily have the same zone representation in storage. There are four zone bits for each character in storage and only two zone positions in a 96-column card column. Therefore a translation must take place when the character is read. The computer checks the entire punch combination (both zone and digit) of a character to determine which bits are turned on or off in order to represent the character in storage.

3. a. 196

```
      1   1  0  0      0  1  0  0
     128+64+0+0   +   0+4+0+0 = 196
```

   b. 213
   c. 232
   d. 243

4. a. C4

```
          1  1  0  0     0  1  0  0
     C = 8+4+0+0   +   0+4+0+0 = 4
```

   b. D5
   c. E8
   d. F3

5. The computer uses the numerical values associated with characters to determine the collating sequence of characters.

6. When characters are collated by zone and digit, they are collated in this order:

Character	Numerical Value
&	80
*	92
/	97
C	195
J	209
P	215
R	217
T	227
0	240
4	244
9	249

When characters are collated by zone the left half of the hexadecimal value is used to detẽrmine the order; when collating by digit the right half of the hexadecimal value is used. When characters are collated by zone or digit, several may hold the same position in the sequence and thus belong in the same group. Within that group they may hold any position.

Characters collated by zone are in this order:

Character	Hexadecimal Value Used
& ⎫ *	50
* ⎭	5C
/	61
C	C3
P ⎫	D7
J ⎬ *	D1
R ⎭	D9
T	E3
4 ⎫	F4
9 ⎬ *	F9
0 ⎭	F0

* Characters within brackets may be in any order since they are in the same group.

Characters collated by digit are in this order:

Character	Hexadecimal Value Used
0 ⎫ *	F0
& ⎭	50
/ ⎫ *	61
J ⎭	D1
C ⎫ *	C3
T ⎭	E3
4	F4
P	D7
R ⎫ *	D9
9 ⎭	F9
*	5C

* Characters within brackets may be in any order since they are in the same group.

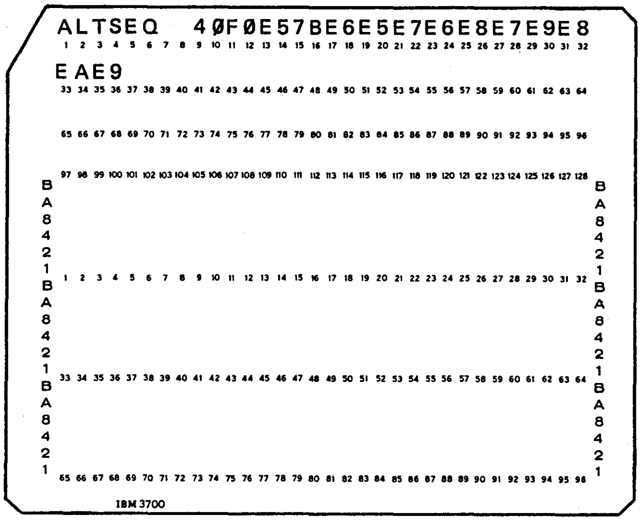
7.

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00110011		33	
00110100		34	
00110101		35	
00110110		36	
00110111		37	
00111000		38	
00111001		39	
00111010		3A	
00111011		3B	
00111100		3C	
00111101		3D	
00111110		3E	
00111111		3F	
01000000	Blank	40	FØ
01000001		41	
01000010		42	
01000011		43	
01000100		44	
01000101		45	
01000110		46	
01000111		47	
01001000		48	
01001001		49	
01001010	¢	4A	
01001011	.	4B	
01001100	<	4C	
01001101	(	4D	
01001110	+	4E	
01001111	|	4F	
01010000	&	50	
01010001		51	
01010010		52	
01010011		53	
01010100		54	
01010101		55	
01010110		56	
01010111		57	
01011000		58	
01011001		59	
01011010	!	5A	
01011011	$	5B	
01011100	*	5C	
01011101	)	5D	
01011110	;	5E	
01011111	¬	5F	
01100000	-	60	
01100001	/	61	
01100010		62	
01100011		63	
01100100		64	
01100101		65	
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011	,	6B	
01101100	%	6C	
01101101	—	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010		7A	
01111011	#	7B	
01111100	@	7C	
01111101		7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	7B
11100110	W	E6	E5
11100111	X	E7	E6
11101000	Y	E8	E7
11101001	Z	E9	E8
11101010		EA	E9
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	



```
ALTSEQ   4ØFØE57BE6E5E7E6E8E7E9E8
EAE9
```

IBM 3700

8. An alternate collating sequence is used only for alphameric compare operations and matching or sequence checking operations done on match fields.

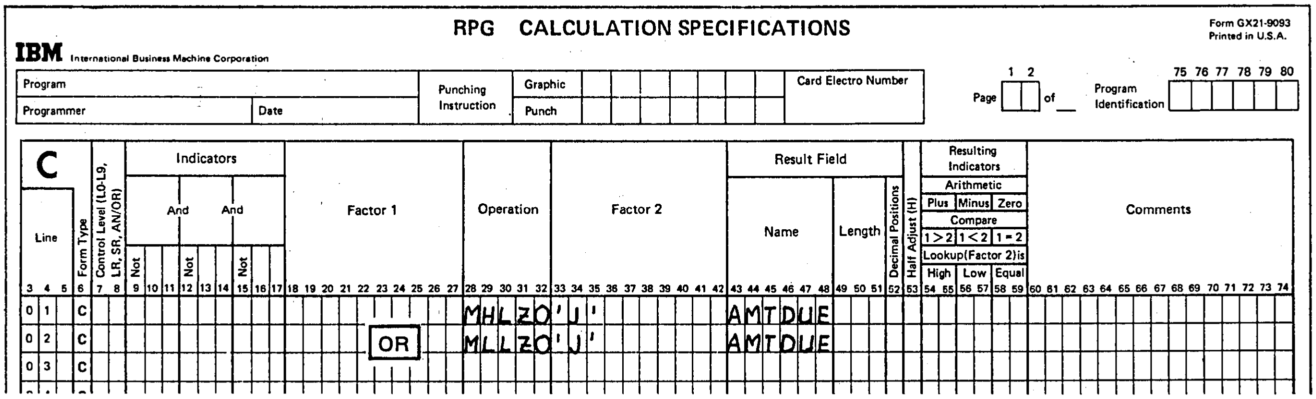
9. The sign of a numeric field must be in the low-order (rightmost) position of the low-order byte.

10. C

11.

	Factor 2	*Result Field*
a.		X
b.	X	
c.	X	X
d.		

12.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C				MHLZO	'J'	AMTDUE					
02	C			OR	MLLZO	'J'	AMTDUE					
03	C											

13. In file translation a character is actually translated into another character because the computer changes bit combinations. All affected characters are changed for the entire program. The bit combinations for characters involved in an alternate collating sequence are not changed. Bit combinations are substituted for others during sequencing operations only.

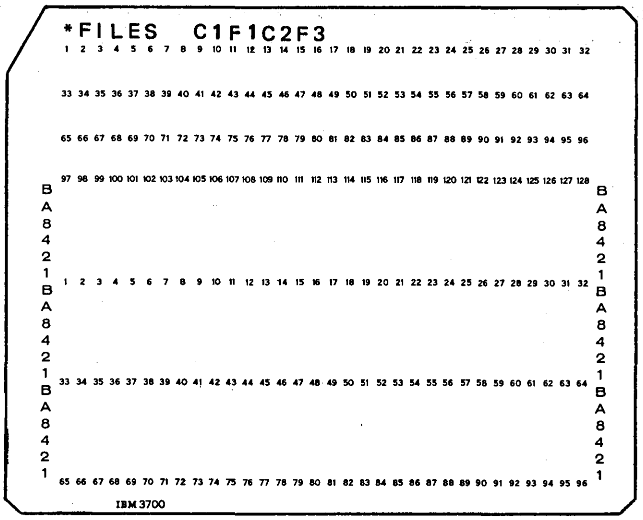
14.

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00110011		33		01100110		66		10011001		99		11001100		CC	
00110100		34		01100111		67		10011010		9A		11001101		CD	
00110101		35		01101000		68		10011011		9B		11001110		CE	
00110110		36		01101001		69		10011100		9C		11001111		CF	
00110111		37		01101010		6A		10011101		9D		11010000	}	D0	
00111000		38		01101011	,	6B		10011110		9E		11010001	J	D1	
00111001		39		01101100	%	6C		10011111		9F		11010010	K	D2	
00111010		3A		01101101	–	6D		10100000		A0		11010011	L	D3	
00111011		3B		01101110	>	6E		10100001		A1		11010100	M	D4	
00111100		3C		01101111	?	6F		10100010		A2		11010101	N	D5	
00111101		3D		01110000		70		10100011		A3		11010110	O	D6	
00111110		3E		01110001		71		10100100		A4		11010111	P	D7	
00111111		3F		01110010		72		10100101		A5		11011000	Q	D8	
01000000	Blank	40		01110011		73		10100110		A6		11011001	R	D9	
01000001		41		01110100		74		10100111		A7		11011010		DA	
01000010		42		01110101		75		10101000		A8		11011011		DB	
01000011		43		01110110		76		10101001		A9		11011100		DC	
01000100		44		01110111		77		10101010		AA		11011101		DD	
01000101		45		01111000		78		10101011		AB		11011110		DE	
01000110		46		01111001		79		10101100		AC		11011111		DF	
01000111		47		01111010	:	7A		10101101		AD		11100000		E0	
01001000		48		01111011	#	7B		10101110		AE		11100001		E1	
01001001		49		01111100	@	7C		10101111		AF		11100010	S	E2	
01001010	¢	4A		01111101	'	7D		10110000		B0		11100011	T	E3	
01001011	•	4B		01111110	=	7E		10110001		B1		11100100	U	E4	
01001100	<	4C		01111111	"	7F		10110010		B2		11100101	V	E5	
01001101	(	4D		10000000		80		10110011		B3		11100110	W	E6	
01001110	+	4E		10000001		81		10110100		B4		11100111	X	E7	
01001111	|	4F		10000010		82		10110101		B5		11101000	Y	E8	
01010000	&	50		10000011		83		10110110		B6		11101001	Z	E9	
01010001		51		10000100		84		10110111		B7		11101010		EA	
01010010		52		10000101		85		10111000		B8		11101011		EB	
01010011		53		10000110		86		10111001		B9		11101100		EC	
01010100		54		10000111		87		10111010		BA		11101101		ED	
01010101		55		10001000		88		10111011		BB		11101110		EE	
01010110		56		10001001		89		10111100		BC		11101111		EF	
01010111		57		10001010		8A		10111101		BD		11110000	0	F0	
01011000		58		10001011		8B		10111110		BE		11110001	1	F1	
01011001		59		10001100		8C		10111111		BF		11110010	2	F2	
01011010	!	5A		10001101		8D		11000000		C0		11110011	3	F3	
01011011	$	5B		10001110		8E		11000001	A	C1	F1	11110100	4	F4	
01011100	*	5C		10001111		8F		11000010	B	C2	F3	11110101	5	F5	
01011101	)	5D		10010000		90		11000011	C	C3		11110110	6	F6	
01011110	;	5E		10010001		91		11000100	D	C4		11110111	7	F7	
01011111	¬	5F		10010010		92		11000101	E	C5		11111000	8	F8	
01100000	-	60		10010011		93		11000110	F	C6		11111001	9	F9	
01100001	/	61		10010100		94		11000111	G	C7		11111010		FA	
01100010		62		10010101		95		11001000	H	C8		11111011		FB	
01100011		63		10010110		96		11001001	I	C9		11111100		FC	
01100100		64		10010111		97		11001010		CA		11111101		FD	
01100101		65		10011000		98		11001011		CB		11111110		FE	
												11111111		FF	



*FILES C1F1C2F3

Chapter 11. DEBUG

CHAPTER 11 DESCRIBES:

How to use the DEBUG operation.

Format of records created by DEBUG.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

RPG II logic for indicators (see *Chapter 1, RPG II Logic*).

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Code the Control Card and Calculation specifications necessary to employ the DEBUG operation.

Place the DEBUG operation within your program so that it will provide meaningful data.

Note: You can use the review questions at the end of this chapter to test your comprehension of this topic. Answers follow the review questions.

INTRODUCTION

A program that you write may not always work perfectly the first time or even the first few times it is run. The reason for this is that the program contains errors — errors that you were not aware you were making when you wrote the program. Some of the errors you can make are easy to find; others may be very difficult to find. Nevertheless, they all have to be corrected. But how do you do this? Where do you start?

Just knowing the types of errors which are commonly made can give you a hint as to what you should check. Most of the errors made fall into one of the following categories:

- Incorrect use of RPG II entries on the specifications sheets.
- Errors in describing input data or the format of output data.
- Errors in specifying the calculation operations.
- Specifying calculation operations in the wrong sequence.

The RPG II Compiler, when compiling your program, will diagnose the specifications to see if they contain errors. If they do, the compiler will print messages telling you the errors made. In this way, you can find errors made in the specifications.

USING THE DEBUG FUNCTION

You may, however, have made all correct entries on the sheets and still get the wrong results. What can you do then? You can, of course, check through your work. But this does not always show you where the error(s) lies.

Sometimes, the specifications you write will not cause the computer to do what you think they will. It is often possible to miss an error because you assume that a statement or group of statements needed to perform a certain task work correctly, when, in fact, they do not. For example, you may pass statement 06 believing it is correct when it is not and spend hours looking for errors in the rest of the statements which are really correct.

It would certainly be helpful to you to find out just how far along in your program everything is working correctly. But how can you find out information your program is working with at various points in your program?

The RPG II language has a special operation code which shows you some of the information the computer is working with. This code is known as the DEBUG code. The code received its name from the slang term "bugs" which is used to mean errors in a program. To debug a program, therefore, means to get all the errors out of it. This is what the DEBUG code helps you do.

The DEBUG code will cause a maximum of two different types of records to be printed or punched out showing you:

- What data is contained in a specified field.
- What indicators are on.

One of the most common errors found in a program is the incorrect use of indicators. If the programmer fails to thoroughly understand RPG II logic, he may condition an operation using an indicator which he thinks is on when it is really not. Thus the program does not work properly.

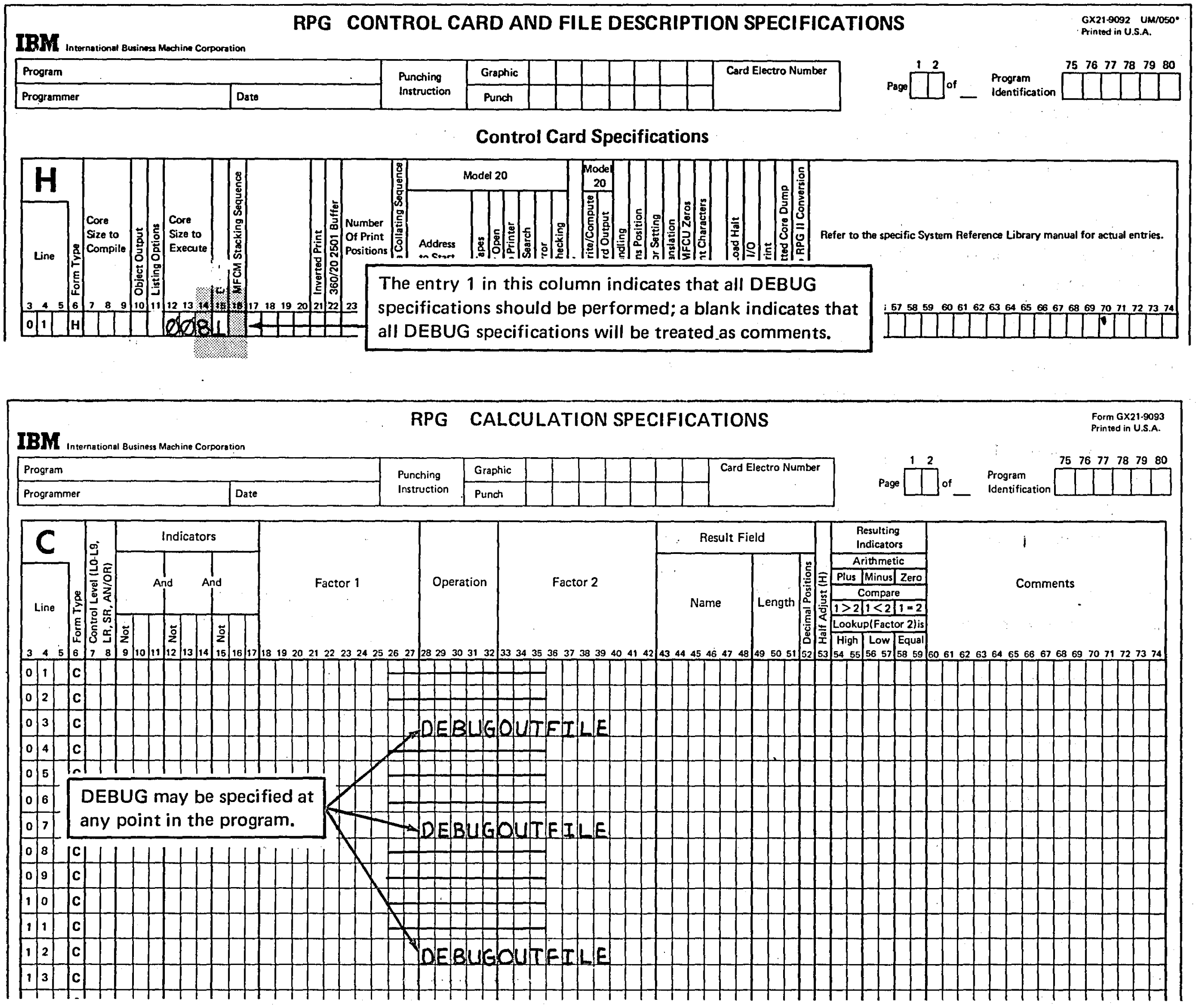
If, at any point in your calculations, you want to check to see if you are using indicators properly you can specify DEBUG. This code will cause a record to be put out showing what indicators are on at the point DEBUG is specified. If you wish to know the contents of a field in addition to knowing what indicators are on, you can also specify so in the DEBUG statement. A second record type will then be put out showing the contents of the field.

Specifications for DEBUG

When using DEBUG, the first specification which you must make is on the control card. A *1* in column 15 indicates that DEBUG is going to be used (see Figure 11-1). If this column is left blank, all DEBUG statements will be treated as comments.

Then in columns 28-32 (Operation) on the Calcultion sheet, specify the code DEBUG. You may specify it at any point in the calculations and as many times as you want.

For each DEBUG statement, enter in columns 33-42 (Factor 2) the name of the file on which DEBUG records will be written or punched (see Figure 11-1). Use the file name previously assigned on the File Description sheet. The same output file must be used for all DEBUG operations in a program.



IBM International Business Machine Corporation

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

GX21-9092 UM/050* Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Control Card Specifications

Line (3-5)	Form Type (6)	Core Size to Compile (7-9)	Object Output (10)	Listing Options (11)	Core Size to Execute (12-14)	(15)
01	H				ØØ8	1

The entry 1 in this column indicates that all DEBUG specifications should be performed; a blank indicates that all DEBUG specifications will be treated as comments.

Refer to the specific System Reference Library manual for actual entries.

IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093 Printed in U.S.A.

Program | Programmer | Date | Punching Instruction | Graphic | Punch | Card Electro Number | Page 1 2 of | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Operation (28-32)	Factor 2 (33-42)
01	C		
02	C		
03	C	DEBUG	OUTFILE
04	C		
05	C		
06	C		
07	C	DEBUG	OUTFILE
08	C		
09	C		
10	C		
11	C		
12	C	DEBUG	OUTFILE
13	C		

DEBUG may be specified at any point in the program.

Figure 11-1. Specifications for DEBUG

The entries just described will give you a record showing what indicators are on. If you also want to know the contents of a field, you must make another entry: The name of the field whose value you wish to know must be entered as the Result Field in Columns 43-48 (see Figure 11-2, insert A).

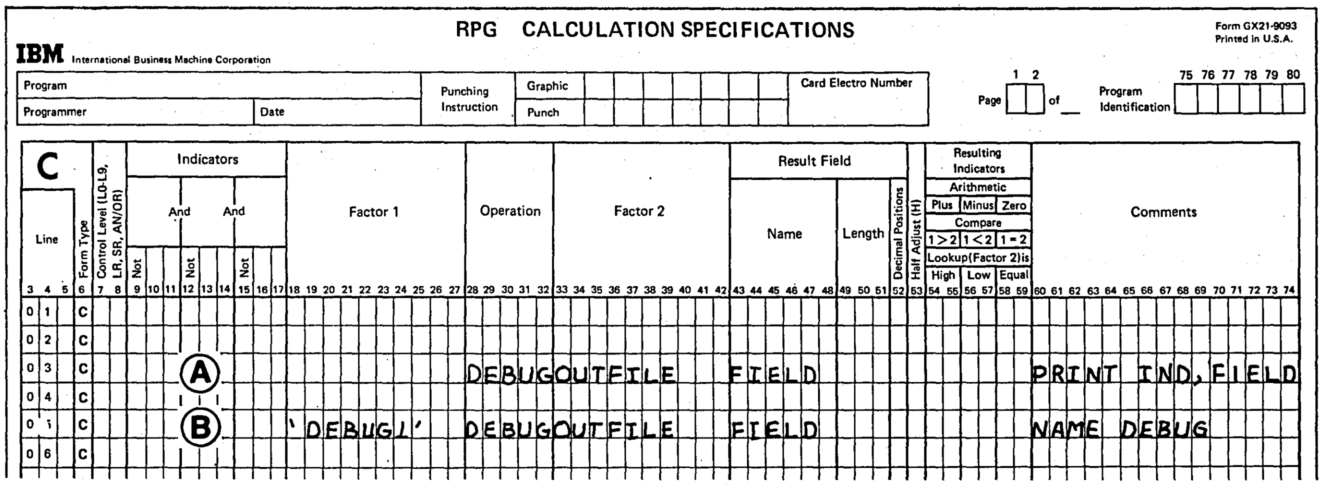
Columns 18-27 (Factor 1) are optional. If you have several DEBUG statements, you may wish to know which records were caused by a particular DEBUG statement. You can name the DEBUG statement by entering a literal in columns 18-27. This name will then be included in the records which the statement causes to be put out (see Figure 11-2, insert B).

Columns 7-17 may contain any valid conditioning indicator. The external indicators U1-U8 are most often used here. They make the DEBUG statement optional. Through their use you can establish, prior to a run, whether or not you wish to use DEBUG (see Chapter 5, *Controlling Operations in an RPG II Program* for uses of U1 to U8). Columns 53-59 cannot be used for the DEBUG statement.

Format of Records Created by DEBUG

Two records may be created by the DEBUG operation. Record 1 is required; Record 2 is optional. Record 1 (Figure 11-3) will look like this:

Positions	*Entry*
1-8	DEBUG =
9-16	The name entered in Factor 1 of the debug statement. These columns will be blank if no entry is found in Factor 1.
17-18	Blank
19-33	INDICATORS ON =
34	Blank
35-37, 38-40, etc.	Name of the indicators which are on. Each indicator is followed by a blank. If a large number of indicators are on, more than one record may be required to show all indicators.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer
Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page □□ of __
Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field Name (43-48)	Length (49-51)	Decimal Positions (52)	Half Adjust (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C											
02	C											
03	C		A		DEBUG	OUTFILE	FIELD					PRINT IND, FIELD
04	C											
05	C		B	'DEBUG1'	DEBUG	OUTFILE	FIELD					NAME DEBUG
06	C											

Figure 11-2. Additional Entries Which Can Be Made for DEBUG

Record 2 (Figure 11-3) will look like this:

Positions	*Entry*
1-14	FIELD VALUE =
15 –	The contents of the field named as the result field in the debug statement. If the field is rather large, only 80 of the rightmost characters are displayed.

Getting Results from DEBUG

DEBUG will not automatically provide you with the specific reason your program is in error. But by showing the indicator setting and contents of fields at various points, it can give you a clue as to where the error lies. From there, you will have to work through the logic of sections in your program to find specific *bugs.*

Placement of DEBUG

DEBUG statements can be placed anywhere in the calculations. However much thought should be given to their position. If they are not placed in proper positions, they may give misleading information and be of no help at all. For example, if you are concerned about the status of an indicator at a certain point in your program, be sure to position DEBUG so that the indicator has no chance to change before it is displayed.

If you want to find out if a statement or group of statements is working correctly you must know what is in the field involved immediately *before* and *after* the statement(s). This means placing DEBUG before and after the statement(s) you are checking. In order to determine if the results obtained from these statements are correct, you will have to manually make the same calculations and compare the two results. In this case, however, you must make certain your own calculations are correct. Much time can be wasted trying to make the computer arrive at the same wrong answer you have calculated.

Making Your Program Work for All Cases

Be certain that you test your program to see that it will handle correctly all possible situations which might arise. If you test for only one or two situations, you can be sure your program works only for those situations. This means that the data you use to test your program be *complete* and valid so that it tests all possible situations. In this way, you can be sure your program can handle all situations without encountering hidden 'bugs'.

Record 1

```
DEBUG = DEBUG 1        INDICATORS ON = 20 42 02 11 MR
```

Record 2

```
FIELD VALUE = 005648219R
```

Figure 11-3. Format of DEBUG Records

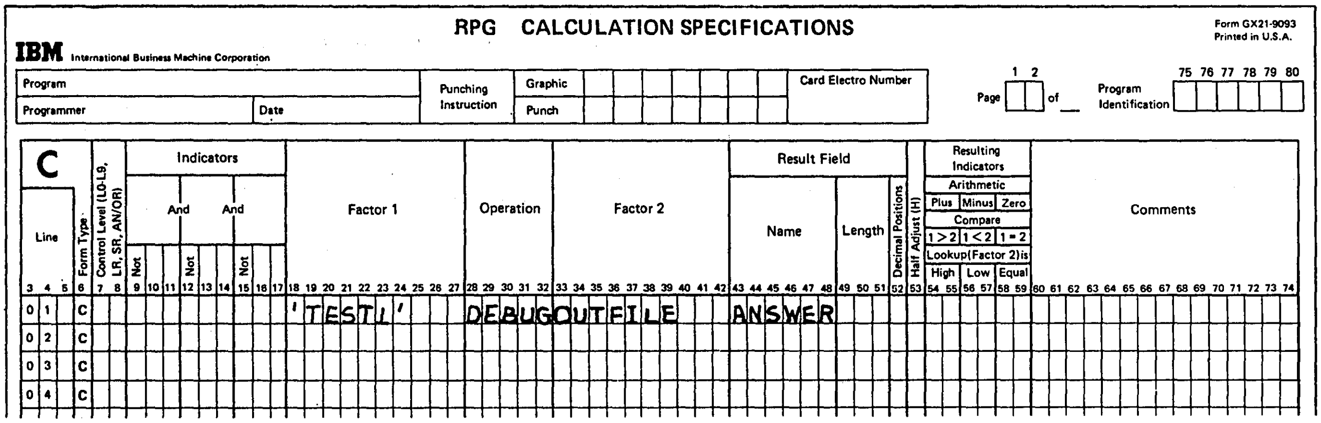
1. What does DEBUG do in an RPG II program?

2. Write a DEBUG statement to display on the file called OUTFILE a field called ANSWER and the indicators that are on at this point. Identify the DEBUG records with the constant - TEST1. What entries in other specification sheets must be made?

Answers To Review 11

1. DEBUG allows you to display the contents of a data field in your program and to list the indicators that are on.

2. A 1 must be entered in column 15 of the RPG II control card.



IBM International Business Machine Corporation

RPG CALCULATION SPECIFICATIONS

Form GX21-9093
Printed in U.S.A.

Program
Programmer Date
Punching Instruction: Graphic / Punch
Card Electro Number
Page 1 2 of
Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR, AN/OR) (7-8)	Indicators: Not, And, Not, And, Not (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field: Name (43-48)	Result Field: Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare 1>2 / Lookup(Factor 2)is High (54-55)	Minus / 1<2 / Low (56-57)	Zero / 1=2 / Equal (58-59)	Comments (60-74)
01	C			'TEST1'	DEBUG	OUTFILE	ANSWER							
02	C													
03	C													
04	C													

Index

READER'S COMMENT FORM

IBM System/3
RPG II Additional Topics
Programmer's Guide

GC21-7567-2

YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page *Comment*

Please include your name and address in the space below if you wish a reply.

● Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

GC21-7567-2

Fold　　　　Fold

Fold　　　　Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS　　PERMIT NO. 40　　ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold　　　　Fold

— Cut Along Line —

IBM

International Business Machines Corporation
General Systems Division
875 Johnson Ferry Road N. E.
Atlanta, Ga. 30342

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

READER'S COMMENT FORM

IBM System/3
RPG II Additional Topics
Programmer's Guide

GC21-7567-2

YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page *Comment*

Please include your name and address in the space below if you wish a reply.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

GC21-7567-2

Cut Along Line

Fold

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

IBM®

International Business Machines Corporation
General Systems Division
875 Johnson Ferry Road N. E.
Atlanta, Ga. 30342

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM System/3 RPG II Additional Tc (File No. S3-28) Printed in U.S.A. GC21-7567-2

GC21-7567-2

IBM System/3 RPG II Additional Topics File No. S3-28 Printed in U.S.A. GC21-7567-2

IBM

International Business Machines Corporation
General Systems Division
875 Johnson Ferry Road, N.E.
Atlanta, Ga. 30342

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)